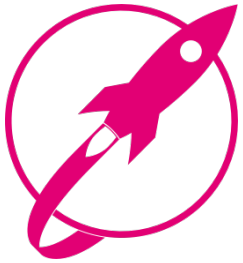


FH·W-S

iOS Programmierung

(mit Swift)

Peter Braun, Florian Bachmann & Andreas Wittmann
@pe_braun @florianbachmann @anwittmann

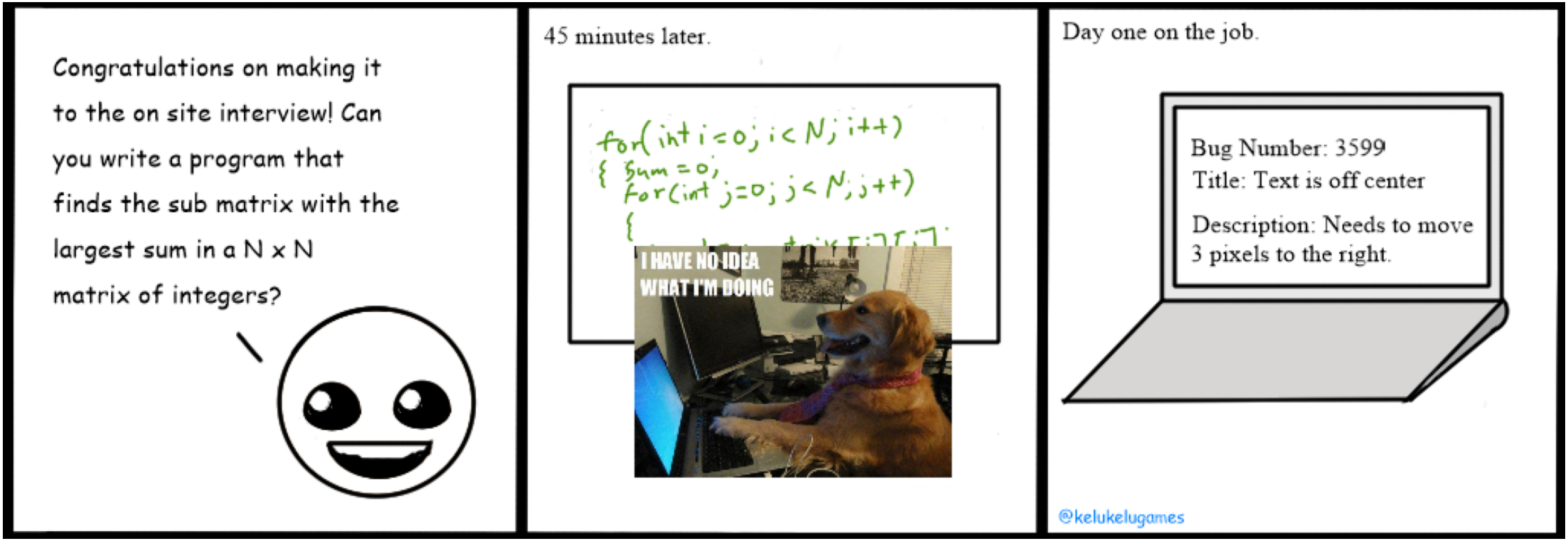
Deutsche Telekom AG
FHWS - Hochschule für angewandte Wissenschaften Würzburg-Schweinfurt

#FHWSSwift

Agenda

1. **Introduction** – Organisatorisches
2. **First iOS-Project** – Hello World, **First iOS-Project** – Still Hello World (now with Code 😄)
3. **Swift**, Wait!, What about Objective-C?, Why Swift?
4. **A (not so) Quick Tour**
5. **Documentation**
6. **The basics –** iOS Architecture & more
7. **User Interfaces –** View Controller, Auto Layout & Size Classes
8. **Storyboard & Segues**
9. **Tables & NavigationController**
10. **TabBarController**
11. **Notifications**
12. **PickerViews**
13. **Touches, Gestures, 3D Touch, Peek & Pop**
14. **ScrollView & StackViews**
15. **Networking** – JSON & Dependency Managers
16. **WebKit**
17. **Maps**
18. **Storage & Data persistency –** NSUserDefaults, NSKeyedArchiver & Core Data
19. **ObjC**

User Interfaces
UIViewController,
Auto Layout & Size Classes

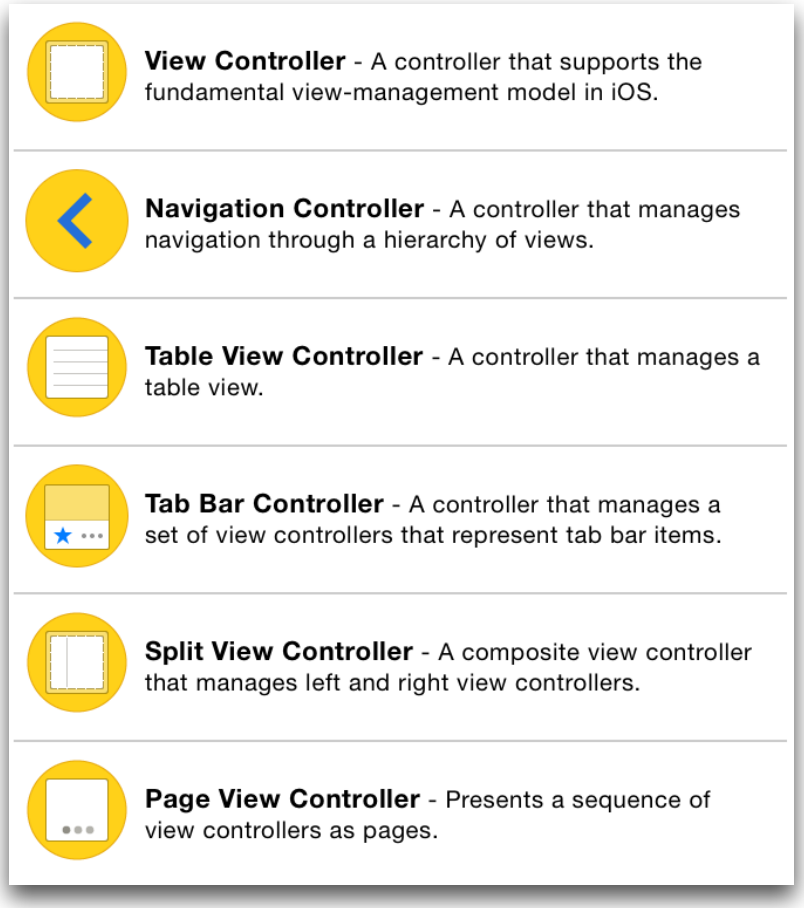


https://twitter.com/KeLuKeLuGames/status/553404094958694400

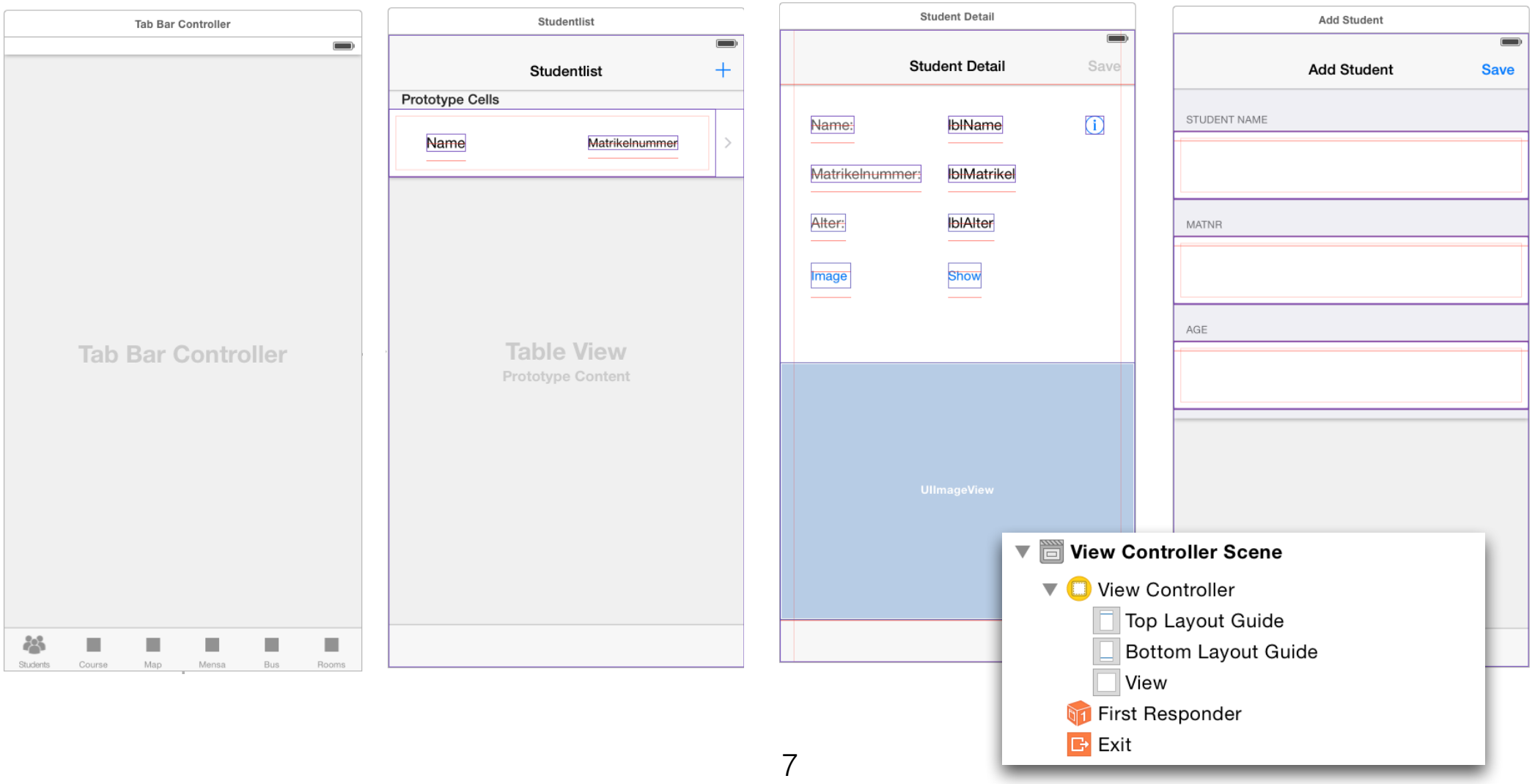
UIViewController

UIViewController

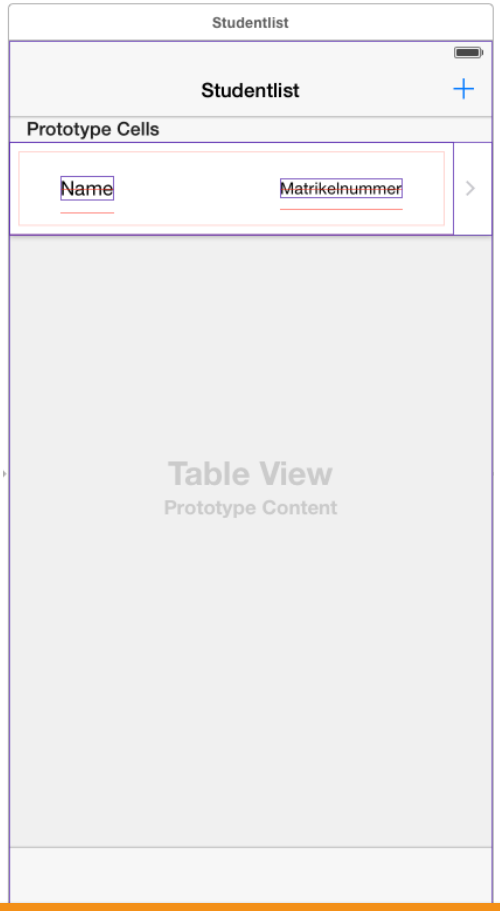
- manages one screen (typically)
  - every screen should have its own controller
- flushes view on low memory situations
- ‘resizes’ its views on orientation changes
- creates modal views on top of current view
- basically loads a ‘nib’



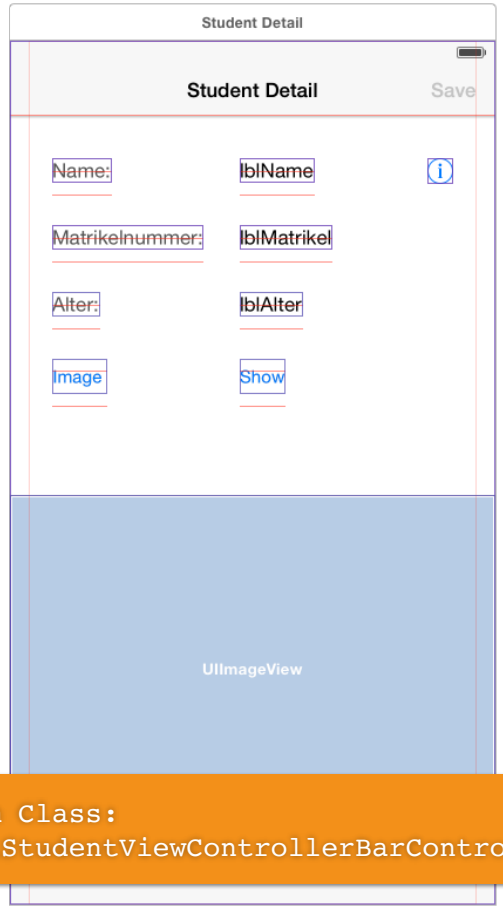
Examples of UIViewControllers



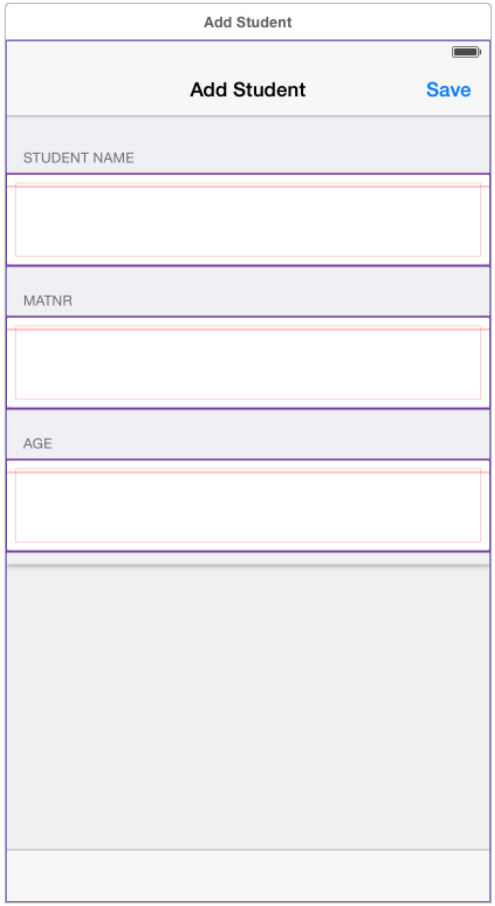
Examples of UIViewControllers



Custom Class:
StudentTableViewControllerBarController



Custom Class:
DetailStudentViewControllerBarController



Custom Class:
AddStudentTableViewControllerBarController

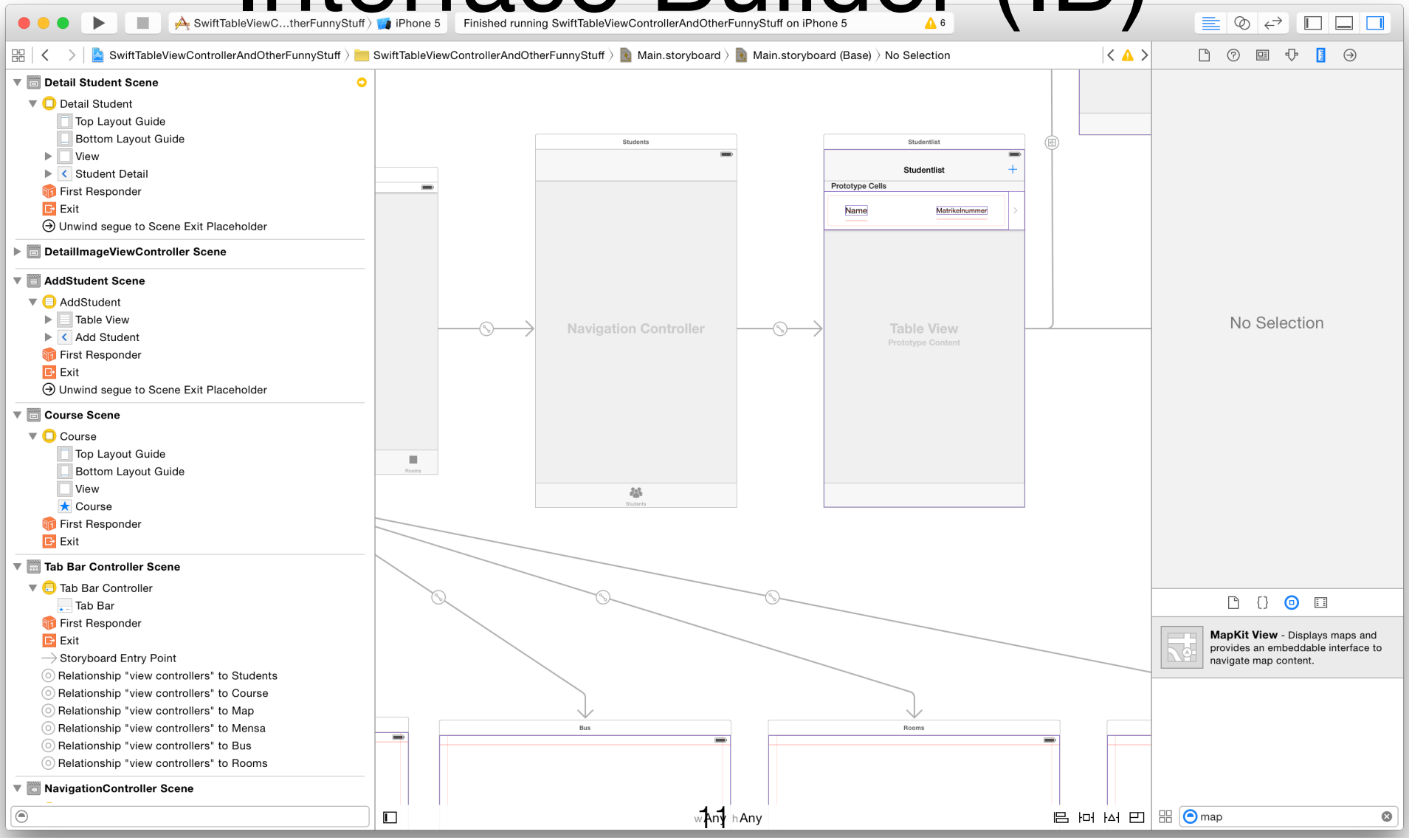
ViewController Life Cycle

UIKit

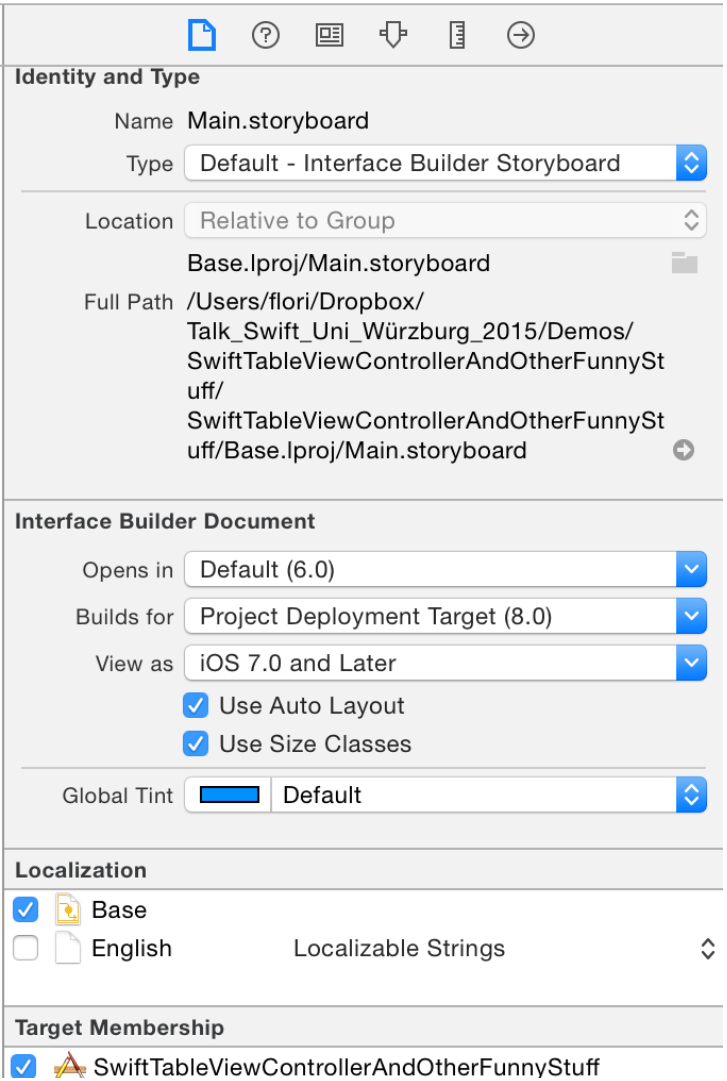
```
init
func loadView()
func viewDidLoad()
func viewWillAppear(animated: Bool)
func viewDidAppear(animated: Bool)
func viewWillDisappear(animated: Bool)
func viewDidDisappear(animated: Bool)
deinit()
```

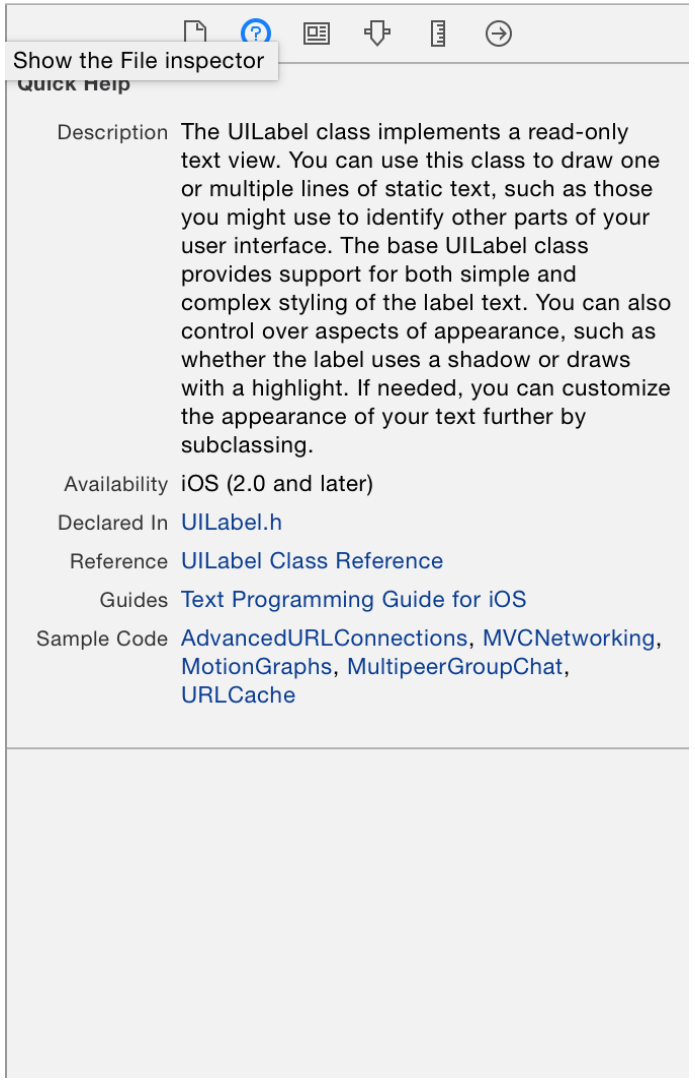
User Interfaces
Auto Layout

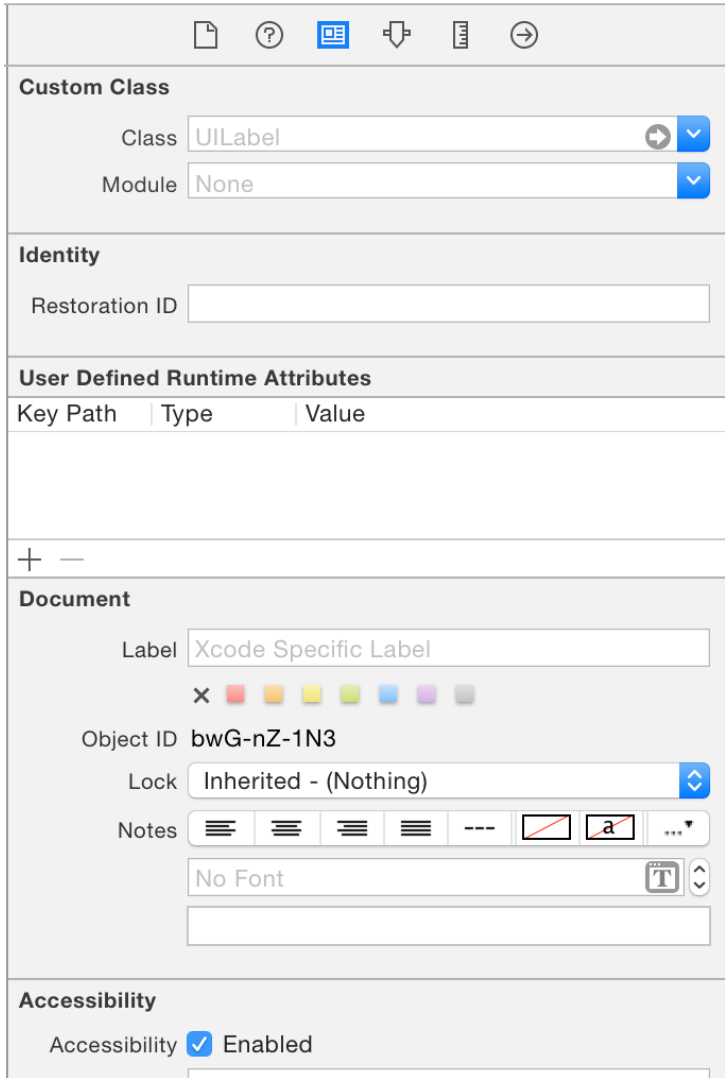
Interface Builder (IB)



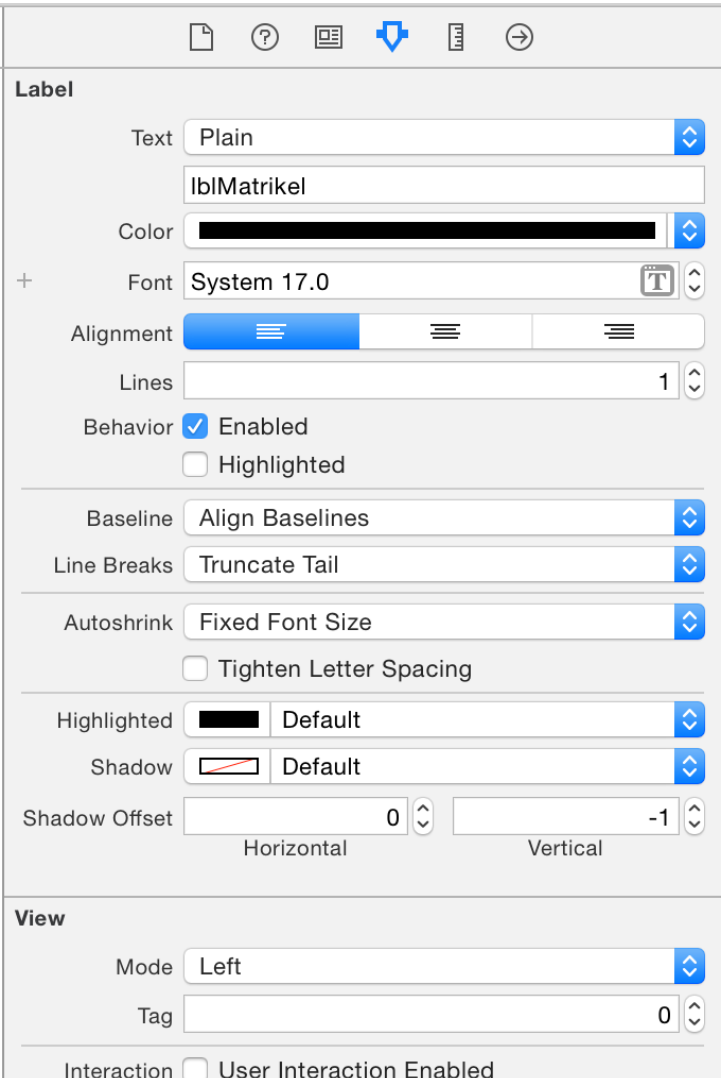
IB - Utilities (1)

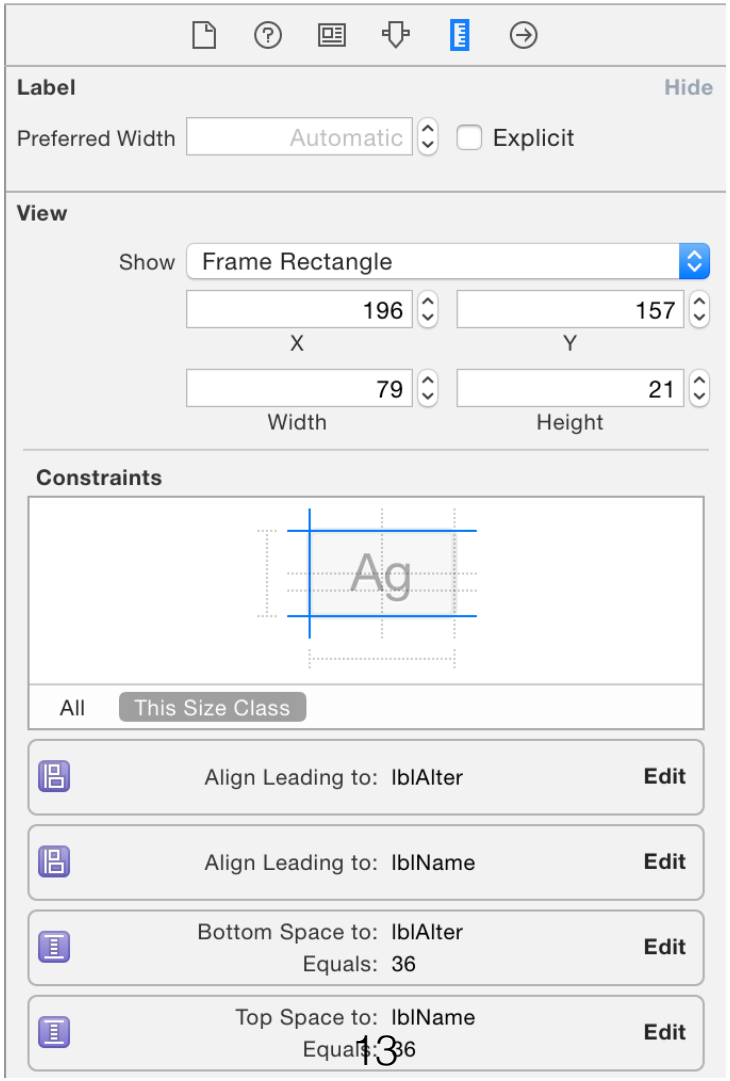


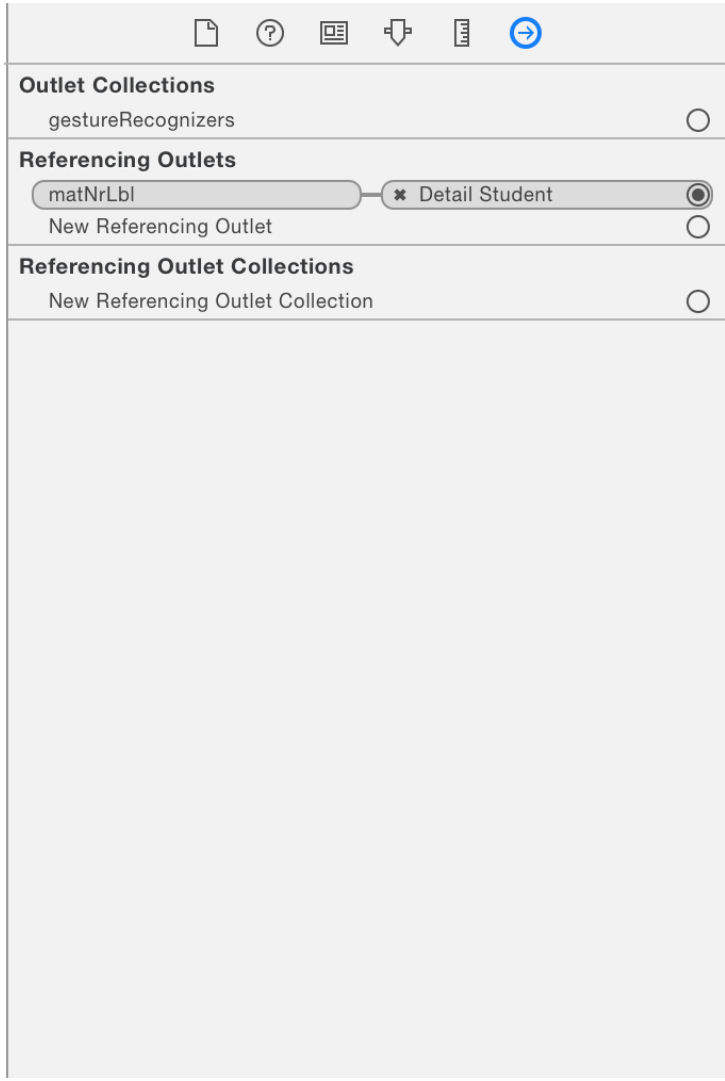




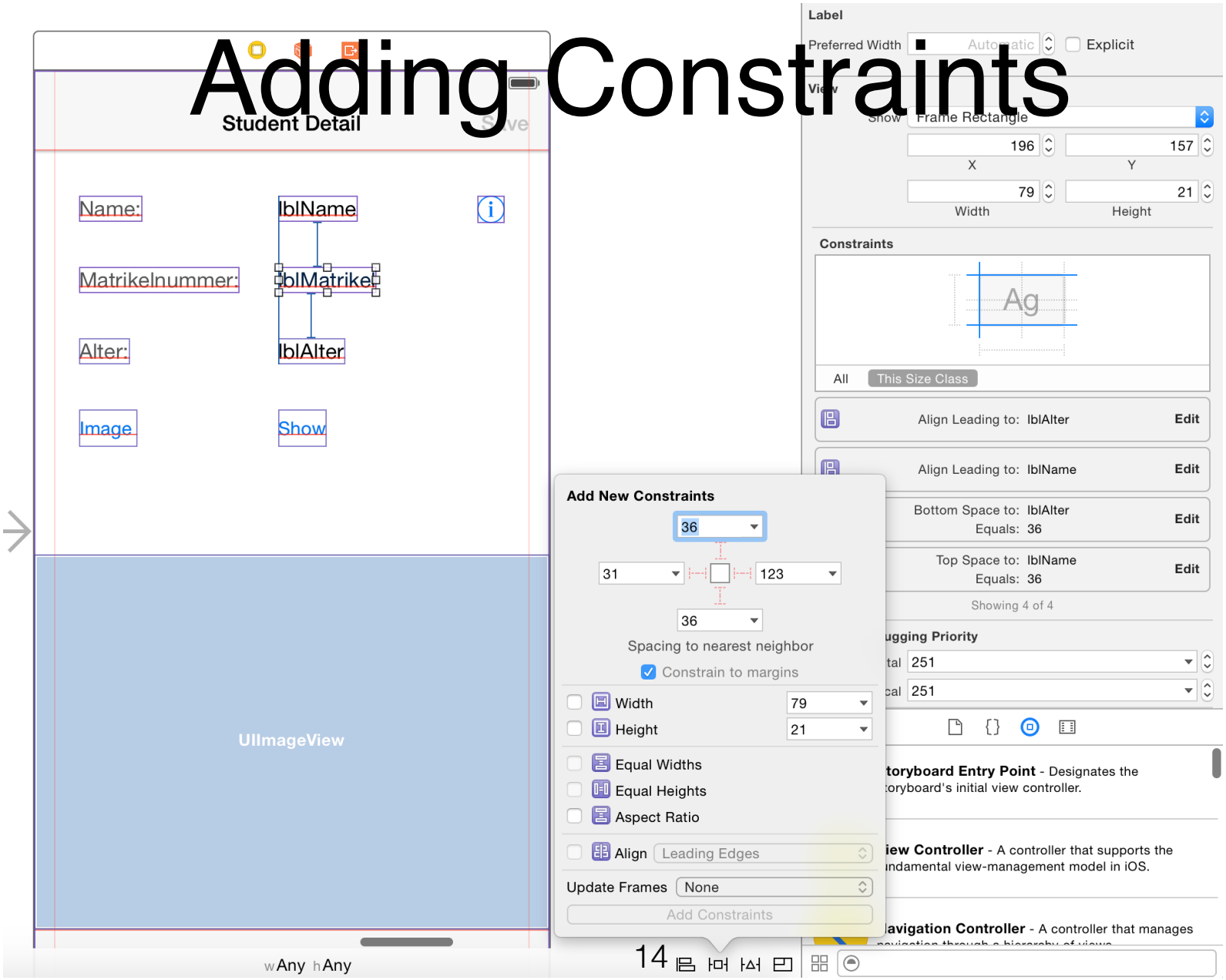
IB - Utilities (2)





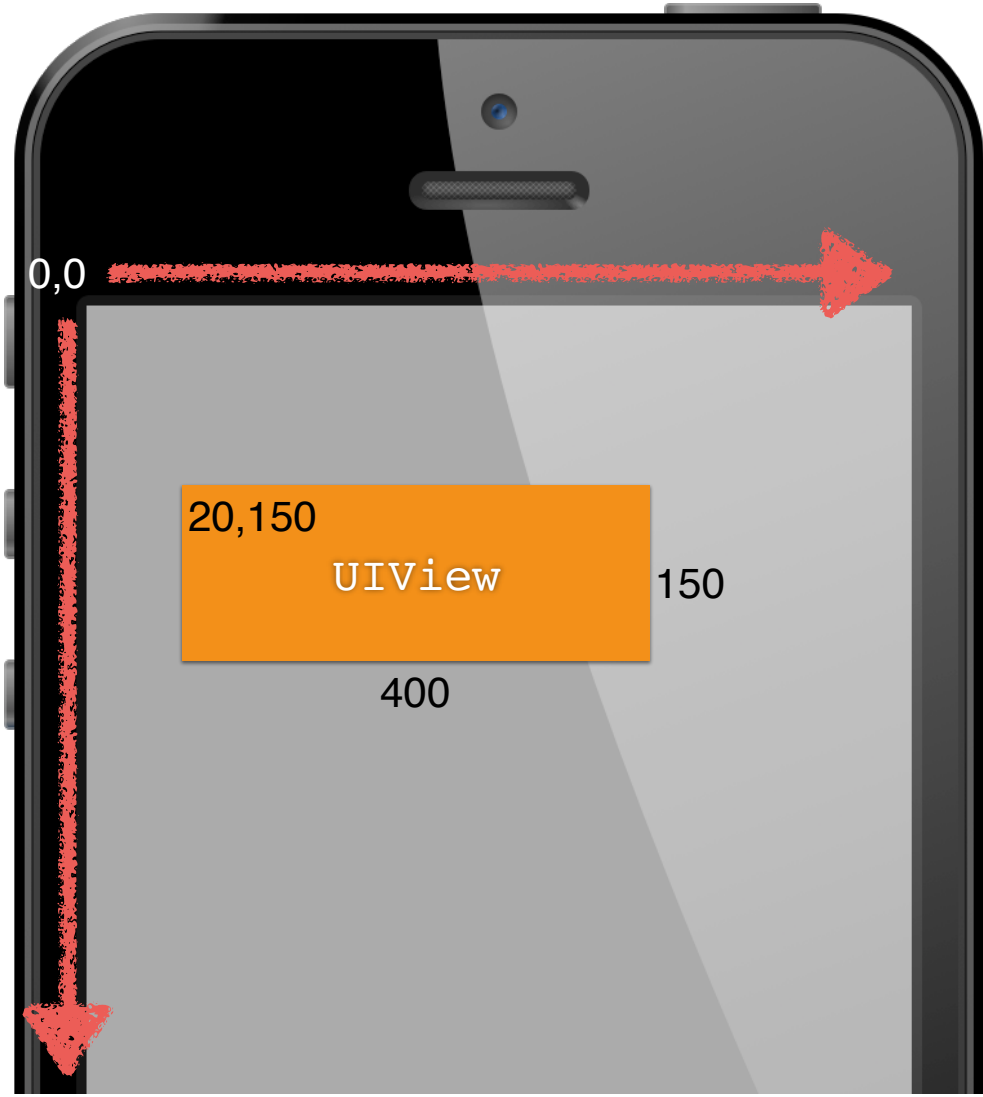


Adding Constraints



Remember - UIView

- defines a rectangular area
- layout as tree
- origin top left
- Frame
  - rectangle with origin & size (relative to superview)
- Bounds
  - rectangle with origin (0,0) and size of view
- Center
  - center point of rectangle

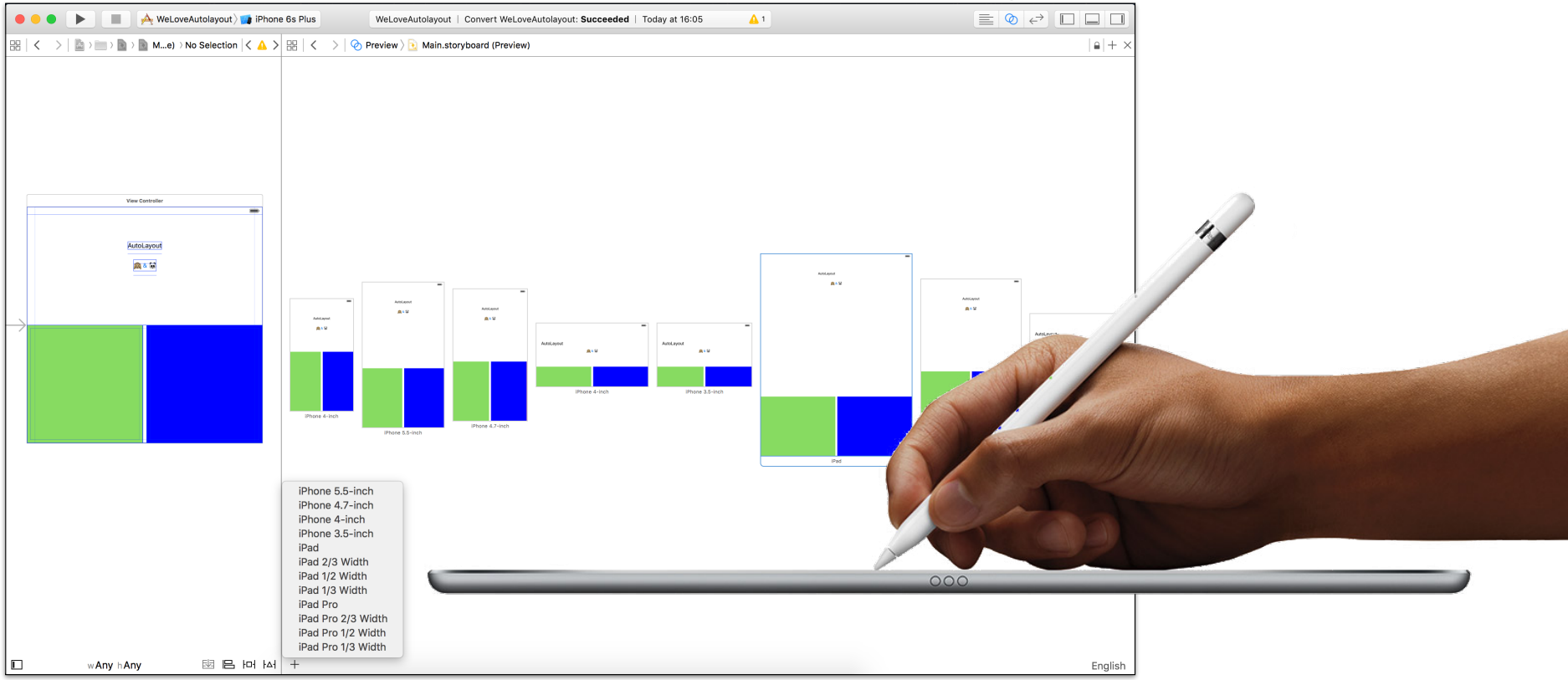


Demo
Auto Layout

User Interfaces
Size Classes / Adaptive Layout

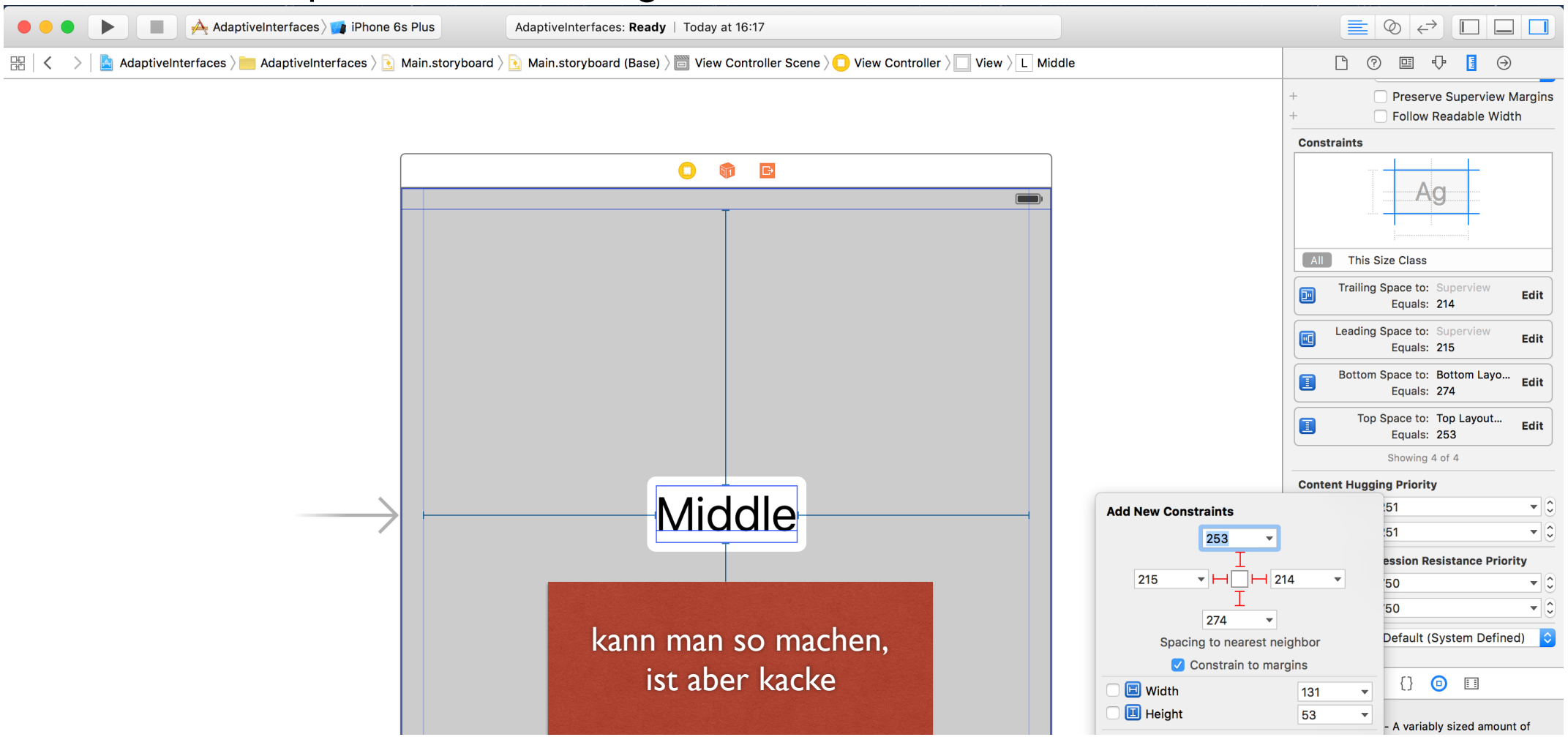
User Interface Possibilities

- Screen sizes, resolutions, and orientations
- at least 12 different form factors (immer noch weniger als bei Android :)



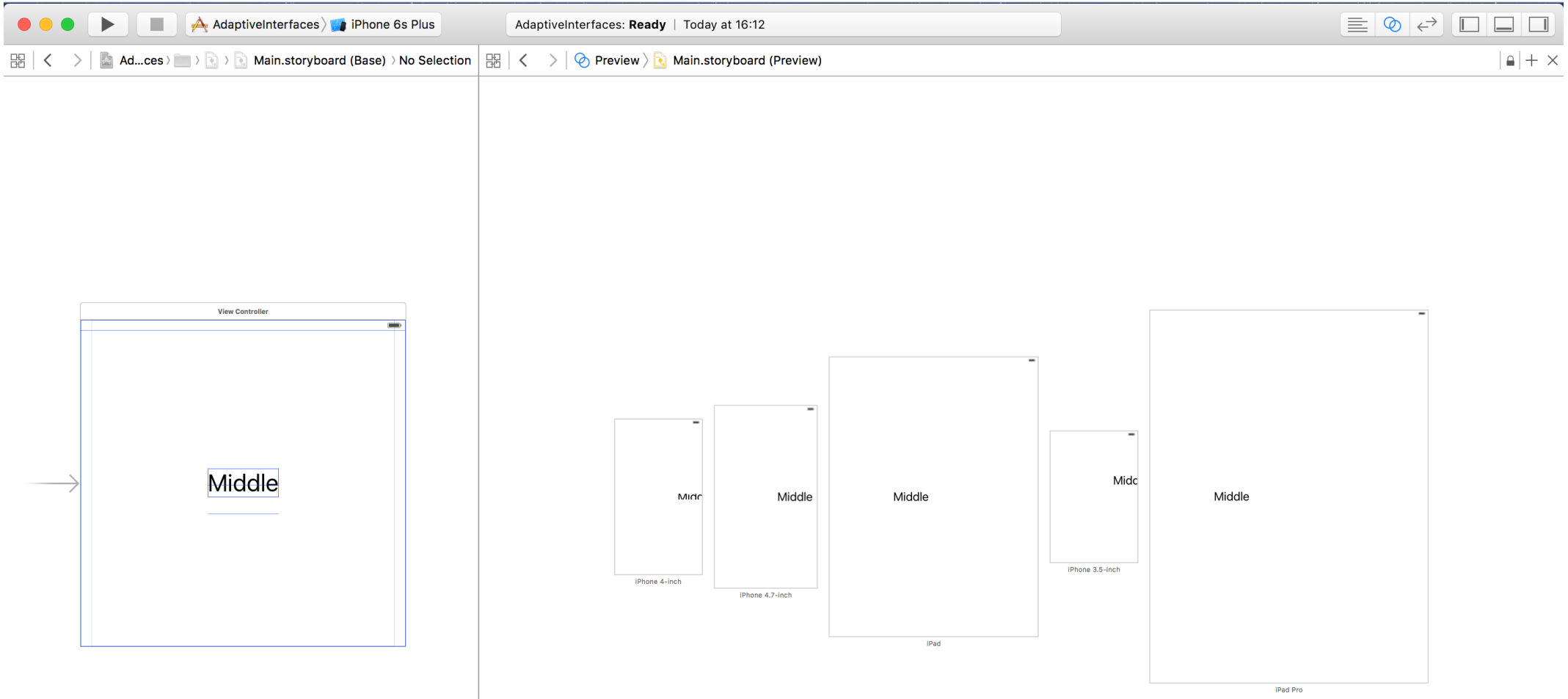
Think adaptive (1/3)

- relative position and sizing



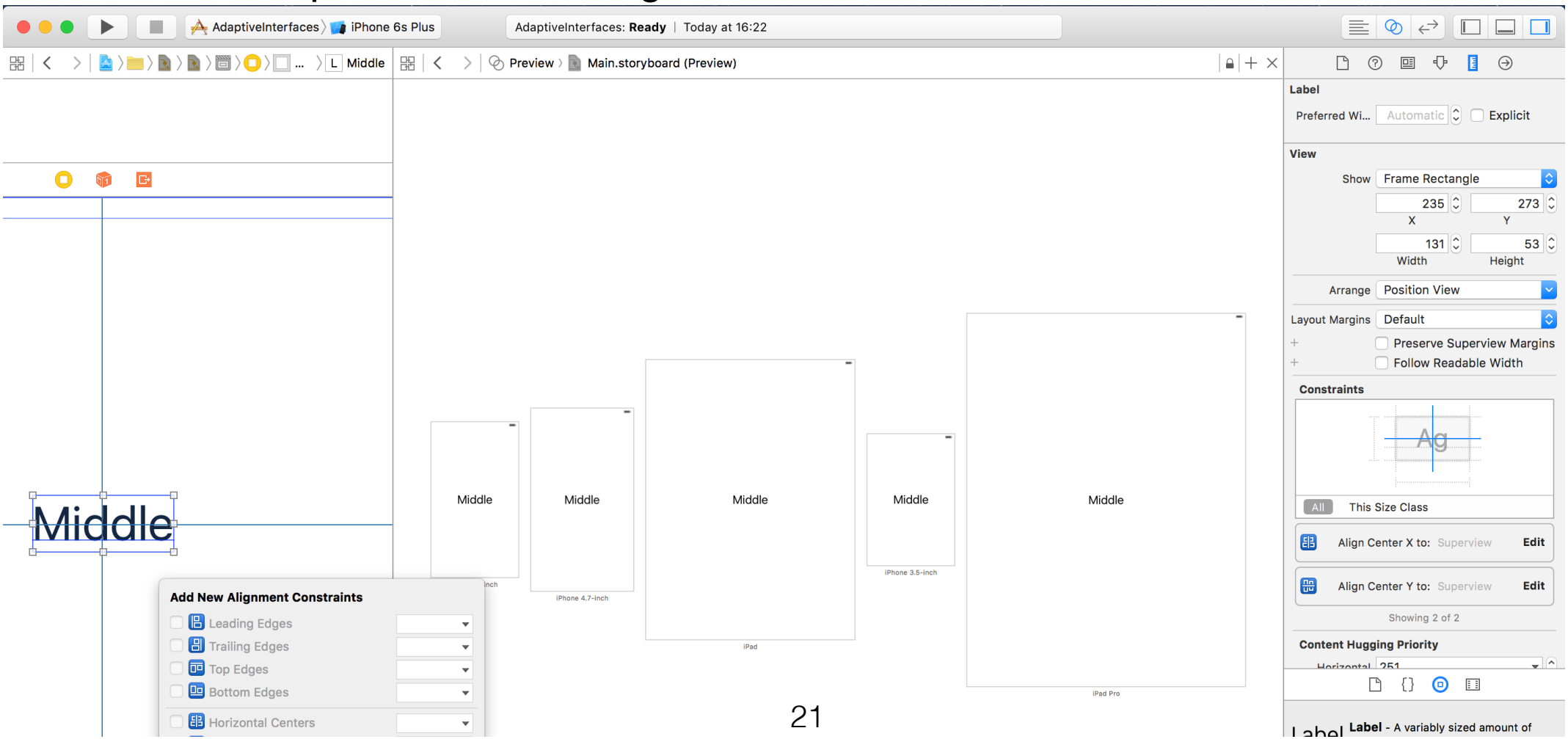
Think adaptive (2/3)

- relative position and sizing



Think adaptive (3/3)

- relative position and sizing



Adaptive Layout

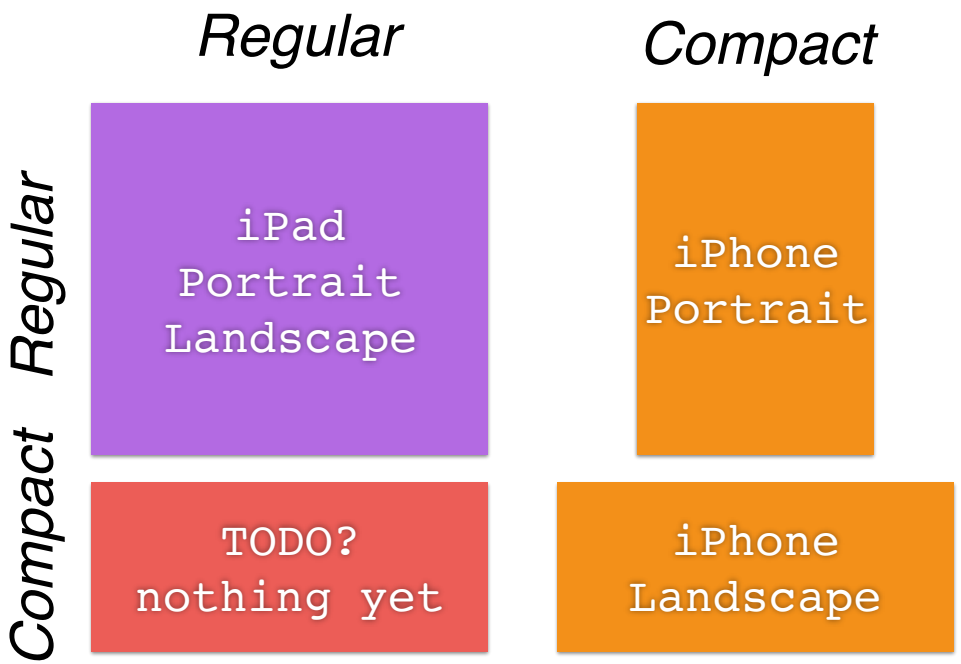
- Abstracts layout away from device specifics
- Introducing concept of size classes
- Available since iOS 8
- fully supported in IB and Xcode

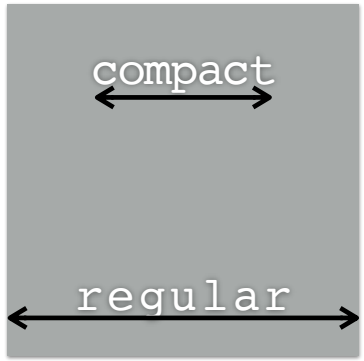
Before Adaptive Layout

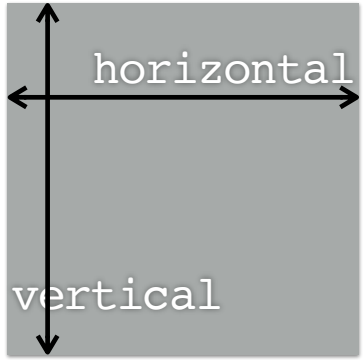
```
let d = UIDevice.currentDevice()
let o = d.orientation
if d.userInterfaceIdiom == .Pad {
    if o == .LandscapeLeft || o == .LandscapeRight {
        doSomethingForIpadsInLandScape()
    } else {
        yeahItIsAnIpadInPortrait()
    }
} else {
    if o == .LandscapeLeft || o == .LandscapeRight {
        itIsAnIphoneInLandscpae()
    } else {
        ohThePhoneIsInPortrait()
    }
}
```

Size Classes (1)

- *Horizontal regular, vertical regular*: iPad in either orientation
- *Horizontal compact, vertical regular*: iPhone portrait
- *Horizontal regular, vertical compact*: no current device
- *Horizontal compact, vertical compact*: iPhone landscape





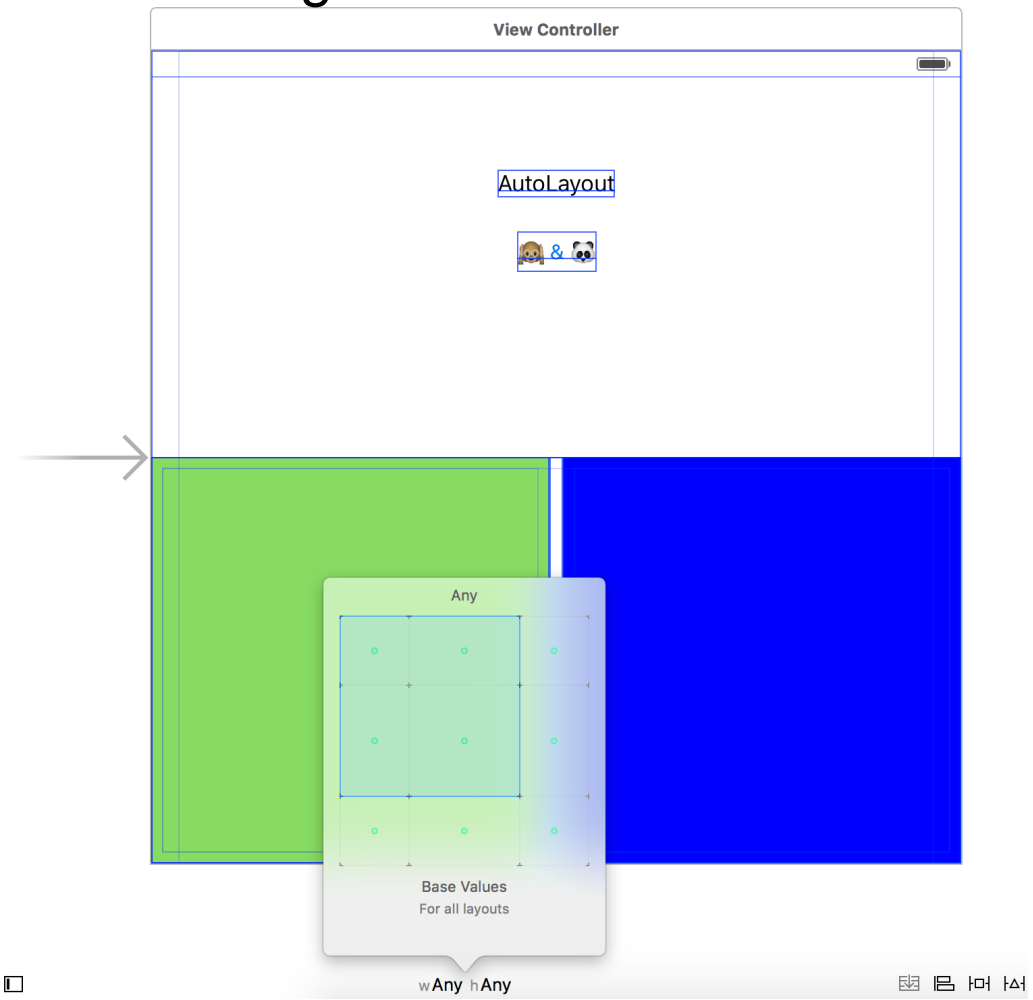


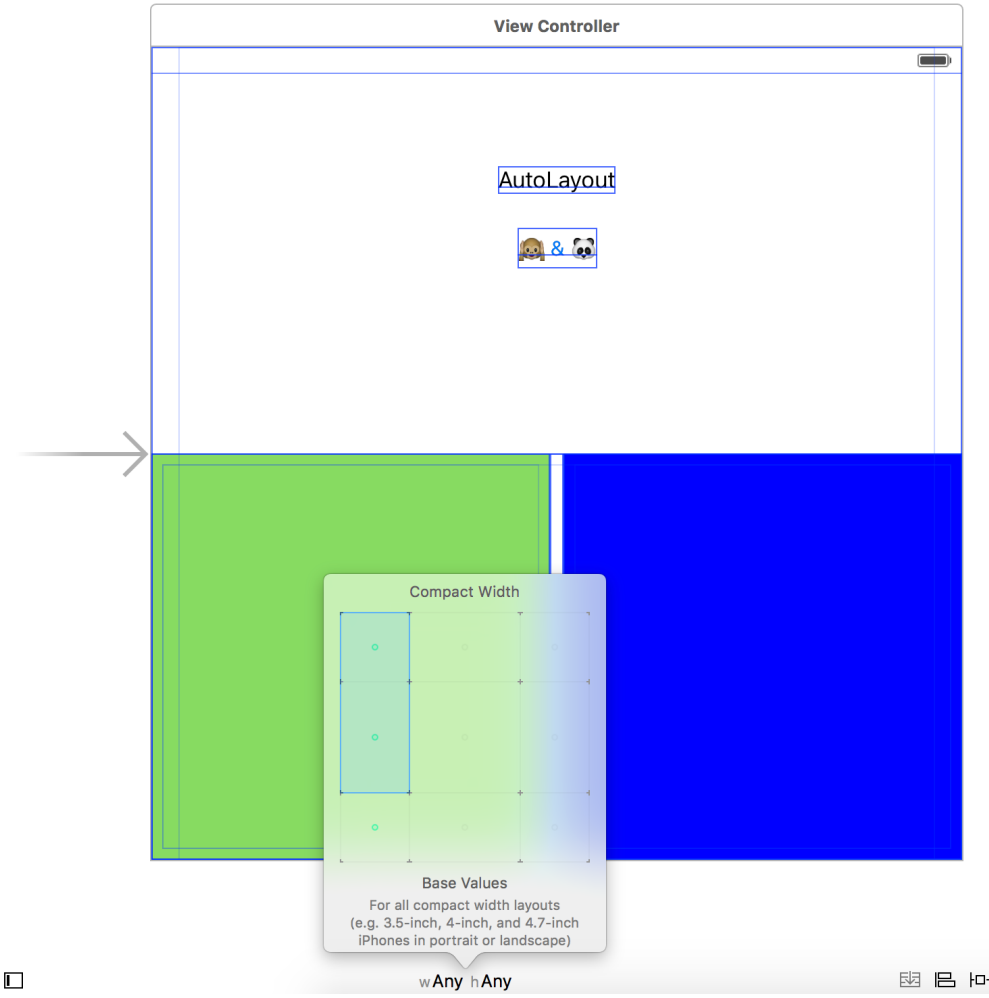
Size Classes (2)

	Vertical Size	Horizontal Size
iPad Portrait	Regular	Regular
iPad Landscape	Regular	Regular
iPhone Portrait	Regular	Compact
iPhone Landscape	Compact	Compact
iPhone 6[s] Plus Landscape	Compact	Regular

Size Classes (3)

thinking abstract about the screens





Size Classes

Specify how objects change between different size magnitudes (size classes)

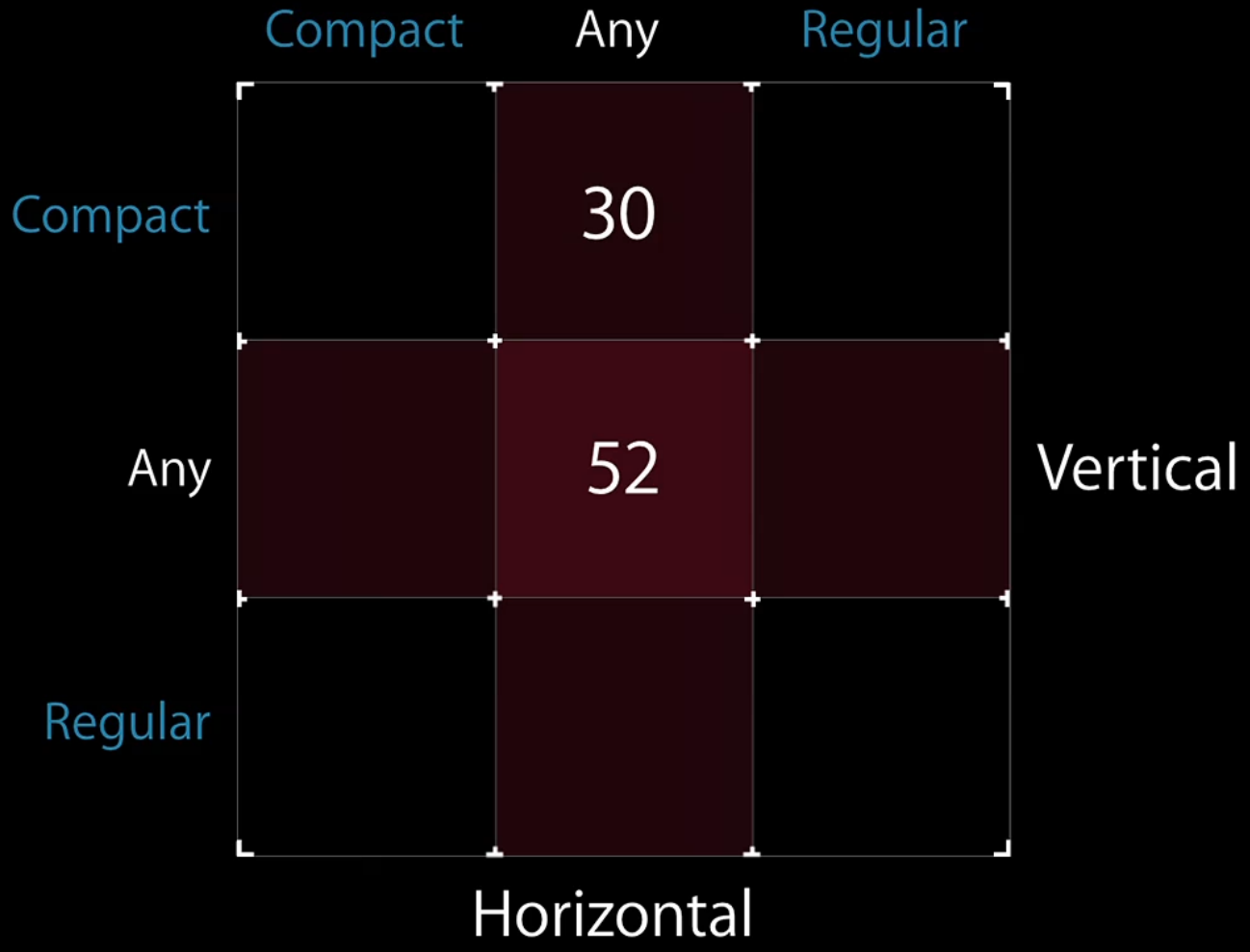
One size class per orientation

Three possible values per size class (Any, Compact, Regular)

Strategy: build in Any | Any, customize in specific configurations where needed

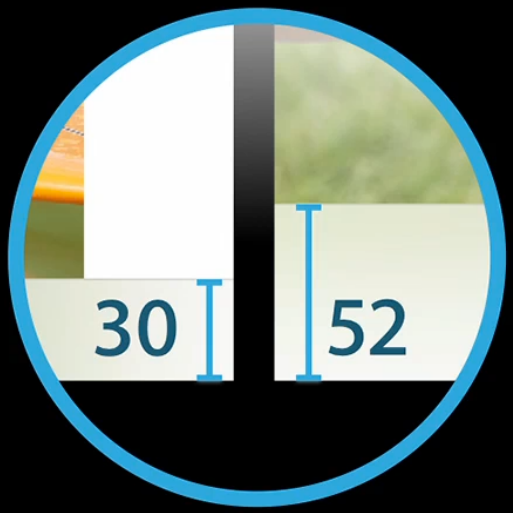
Size Classes

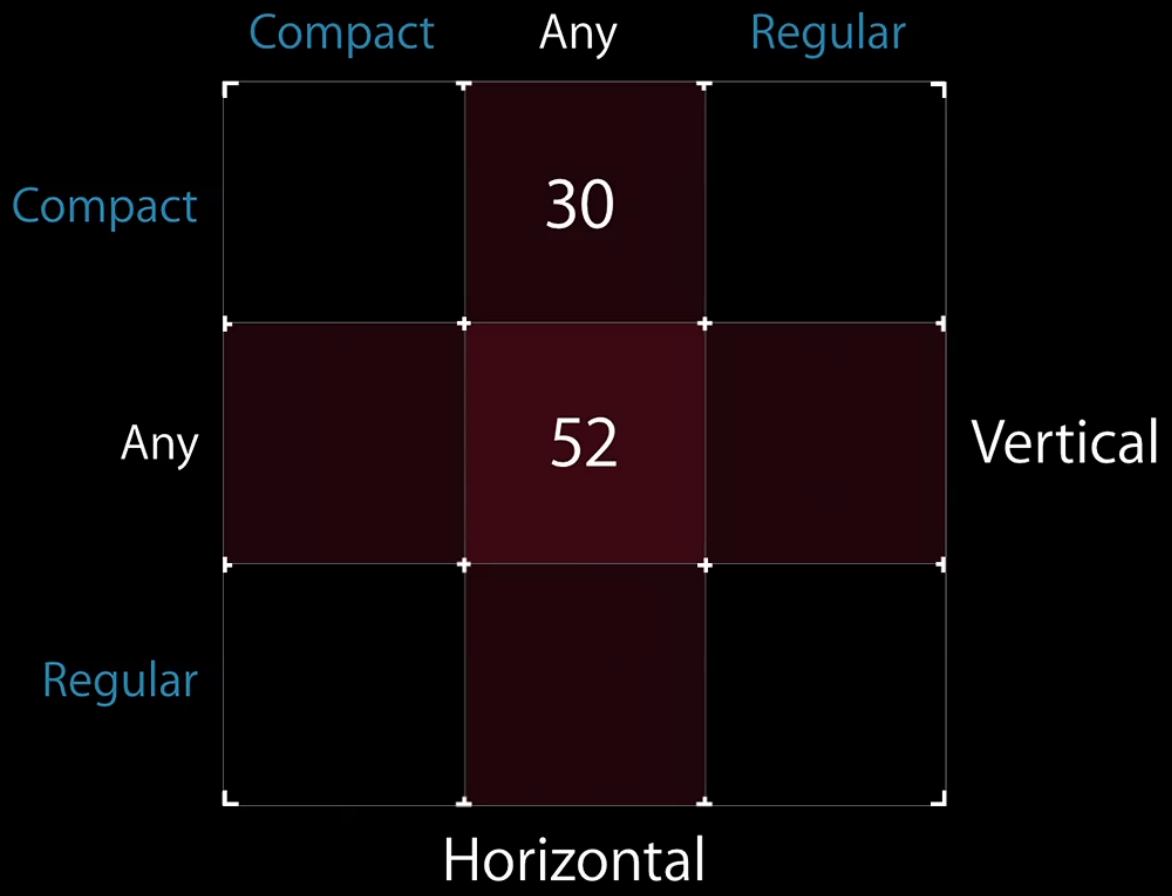
Example: Adapting the bar height



Size Classes

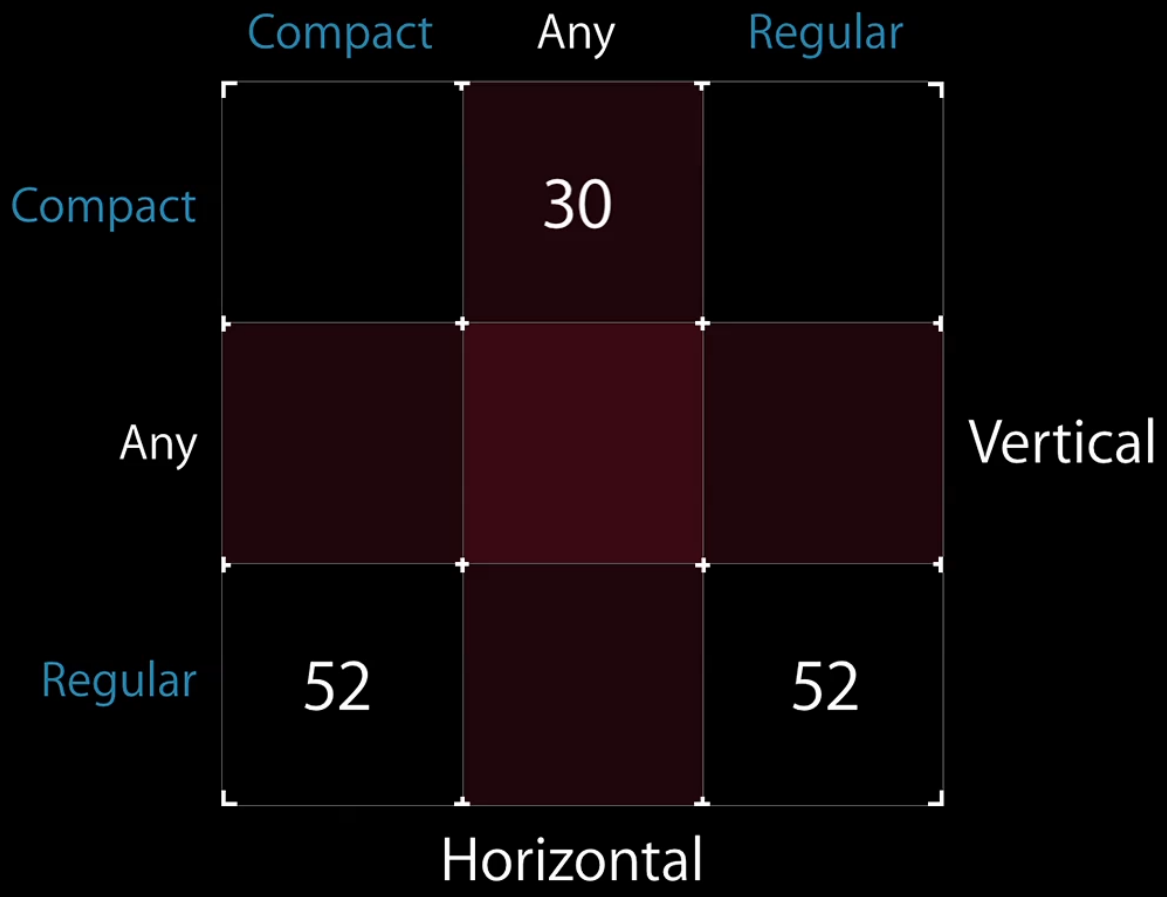
Example: Adapting the bar height





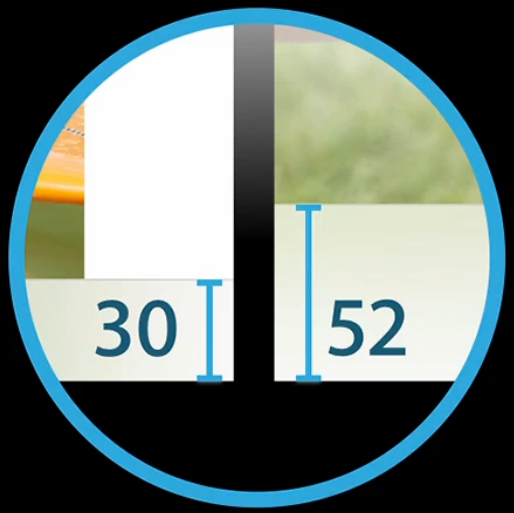
Size Classes

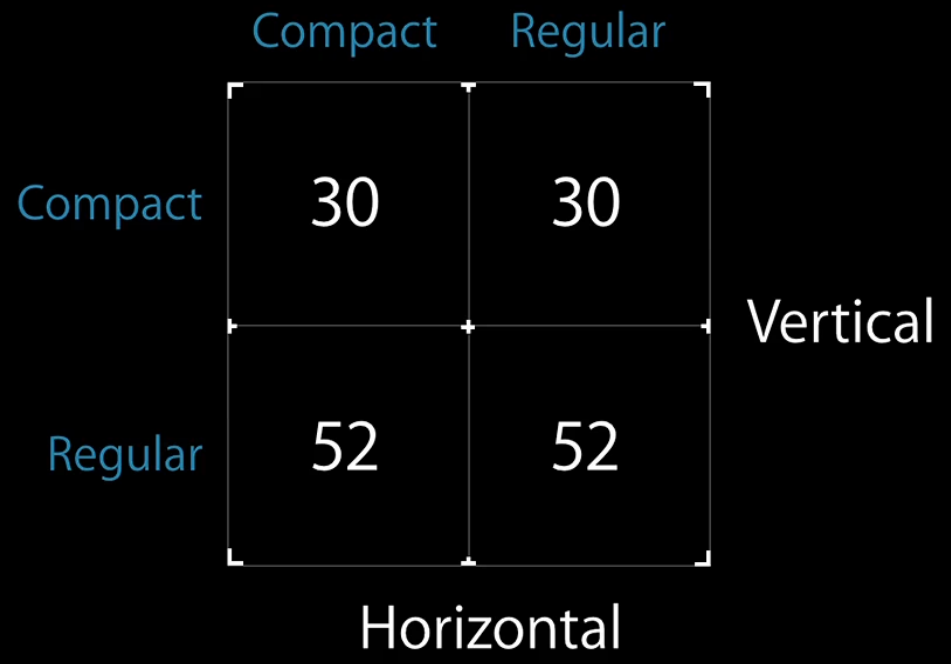
Example: Adapting the bar height



Size Classes

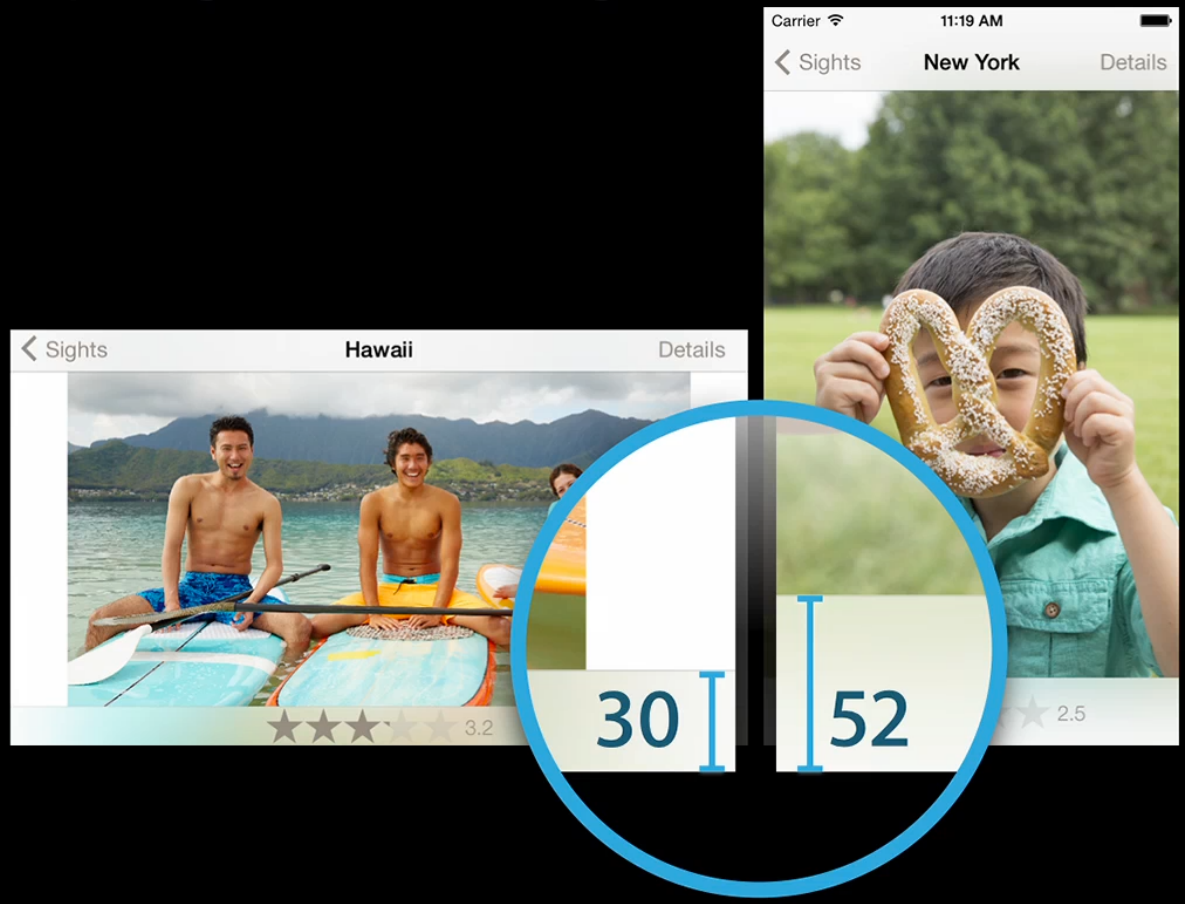
Example: Adapting the bar height





Size Classes

Example: Adapting the bar height



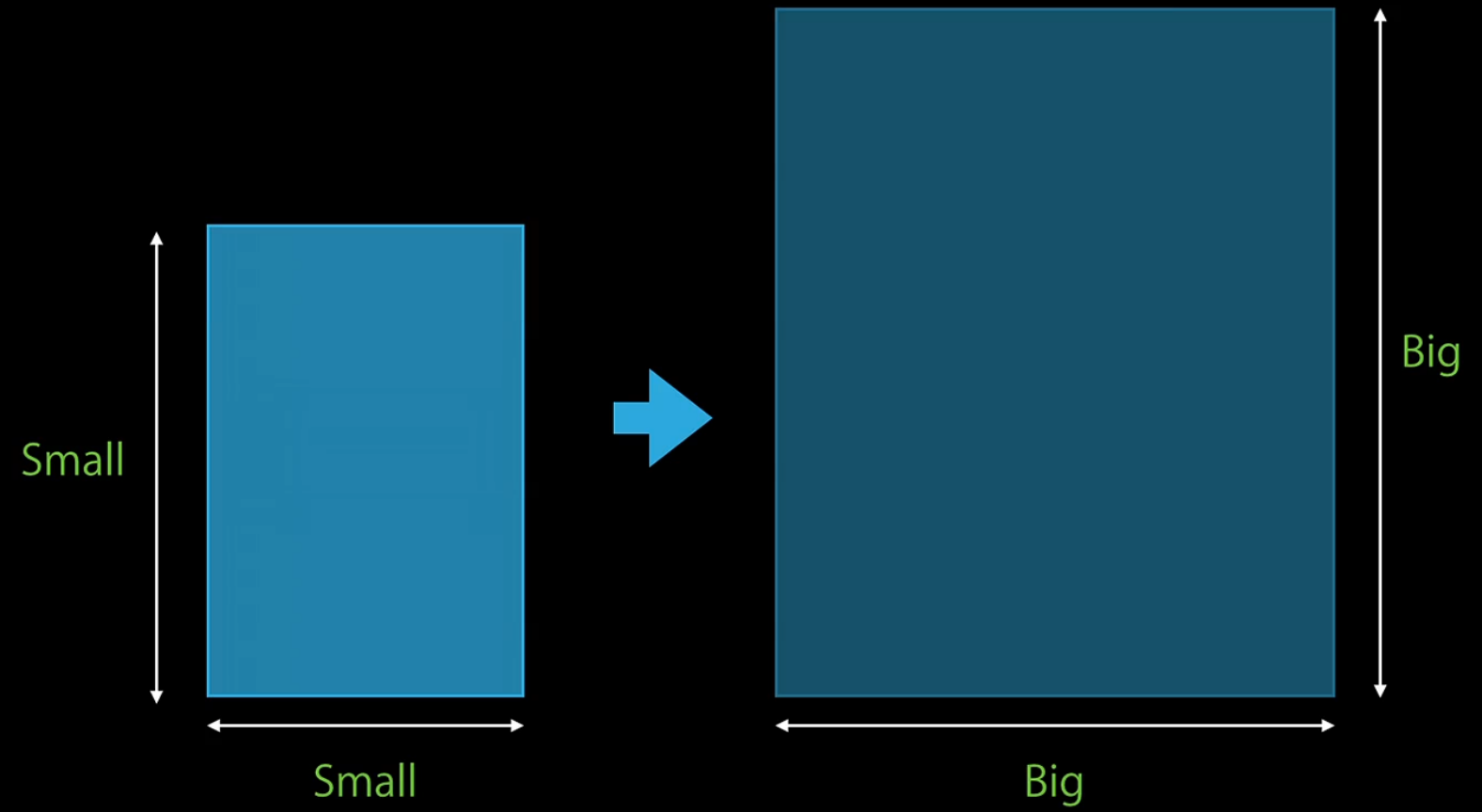
Size Classes

Search Results

Values	Compact	Regular
Orientation	Horizontal	Vertical

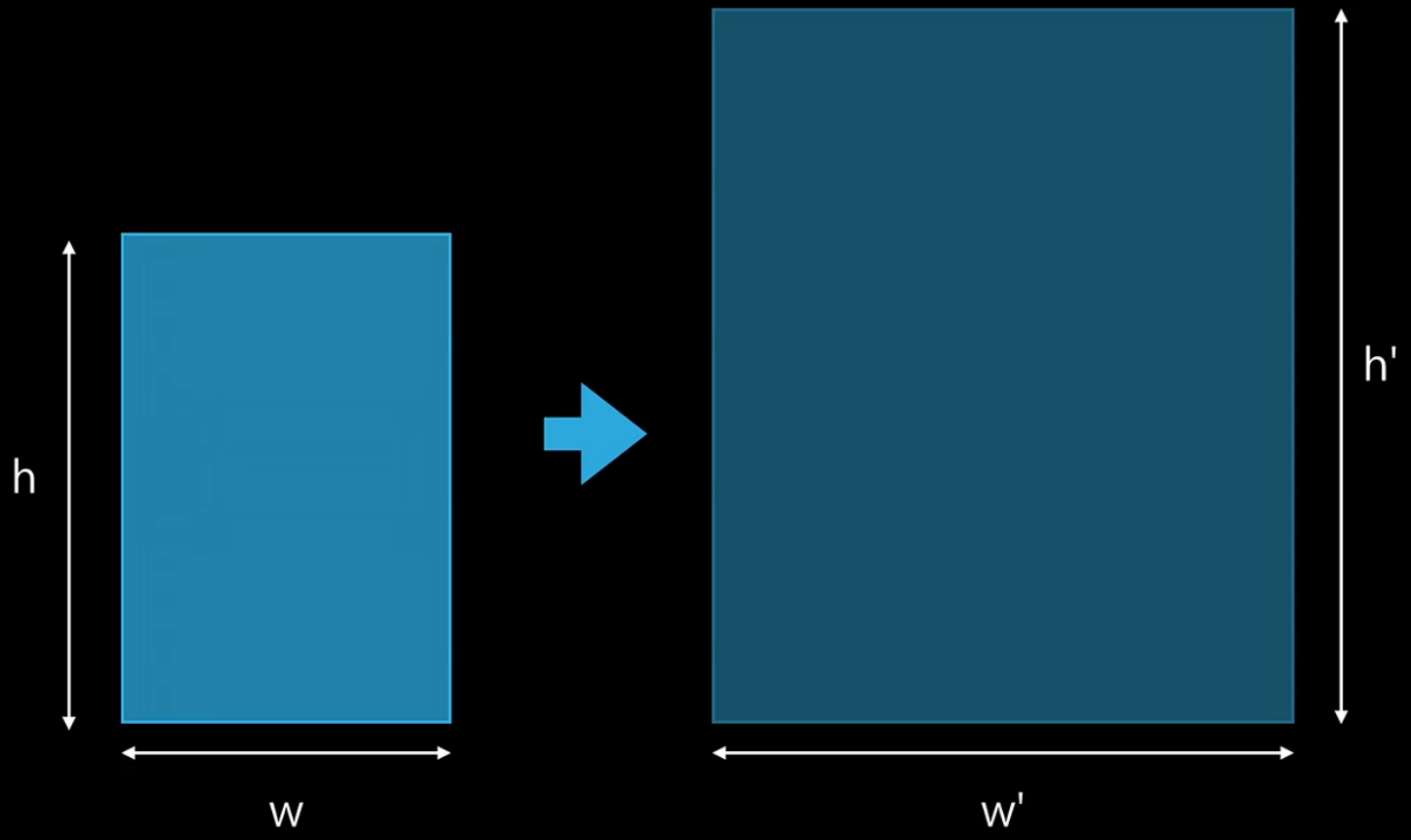
Size Classes

Thinking in terms of magnitude



Size Classes

Thinking in terms of magnitude



Allows you to specify how your UI changes when the available size of your view controller changes

Start with
Base Layout
and then
override

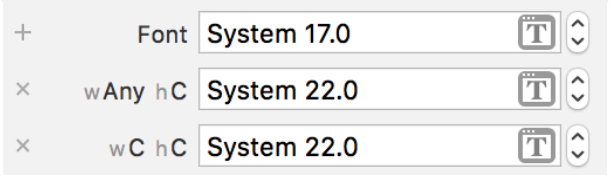
Adopting Adaptive Layout

1. *Build base layout*
2. *Choose size class override*
3. *uninstall irrelevant constraints*
4. *add new constraints specific to size class*
5. *name important constraints (with emojis ↔️, ↕️&✅)*
6. *repeat*

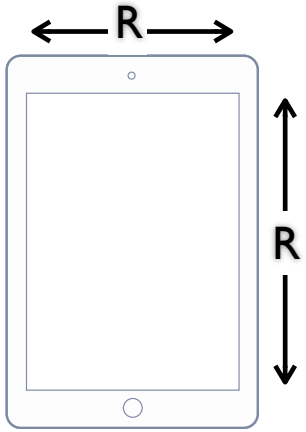
Adaptivity is:

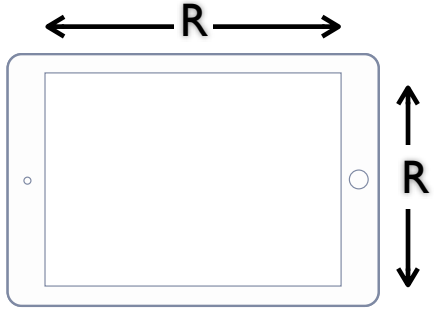
- *constraint installation*
- *constraint constant*
- *font*
- *view installation*

(Basically everything with a plus symbol)

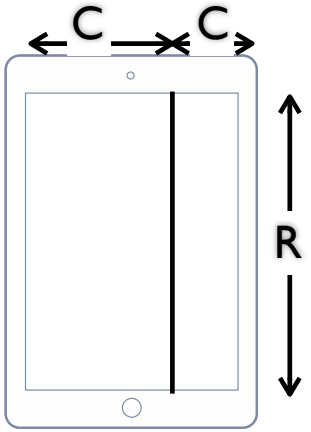


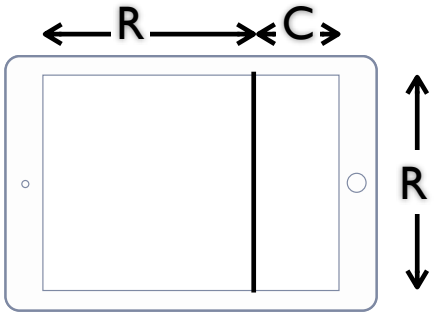
Size Classes - Multitasking

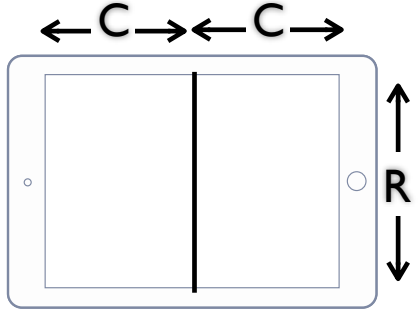




C = compact
R = regular







UITraitEnvironments

Trait environments expose a trait collection that describes their environment.

```
public protocol UITraitEnvironment : NSObjectProtocol {
    public var traitCollection: UITraitCollection { get }

    public func traitCollectionDidChange(previousTraitCollection: UITraitCollection?)
}
```

- UIScreen
- UIWindow
- UIPresentationController
- UIViewController
- UIView

UITraitCollection

A trait collection encapsulates the system traits of an interface's environment.

```
public protocol UITraitEnvironment : NSObjectProtocol {
    public var traitCollection: UITraitCollection { get }

    public func traitCollectionDidChange(previousTraitCollection: UITraitCollection?)
}
```

- UIScreen
- UIWindow
- UIPresentationController
- UIViewController
- UIView

UIContentContainer
React to changes in Code

```
public protocol UIContentContainer : NSObjectProtocol {

… systemLayoutFittingSizeDidChangeForChildContentContainer(
      container: UIContentContainer
  )

… viewWillTransitionToSize(
      size: CGSize,
      withTransitionCoordinator coordinator: UIViewControllerTransitionCoordinator
  )

… willTransitionToTraitCollection(
      newCollection: UITraitCollection,
      withTransitionCoordinator coordinator: UIViewControllerTransitionCoordinator
  )
```

UIContentContainer Rotation - Example

```
…

override func viewWillTransitionToSize(
      size: CGSize,
      withTransitionCoordinator coordinator: UIViewControllerTransitionCoordinator
  ) {
   super.viewWillTransitionToSize(size, withTransitionCoordinator: coordinator)
   let image = imageForAspectRatio(size.width / size.heigh)

   coordinator.animateAlongsideTransition({
     context in
     let transition = CATransition()
     transition.duration = context.transitionDuration()
     transition.timingFunction = CAMediaTimingFunction(…: .EaseInEaseOut)
     self.backgroundImageView.layer.addAnimation(transition.duration, forKey: “Fade”
     self.backgroundImageView.image = image
}, completion: nil)
```

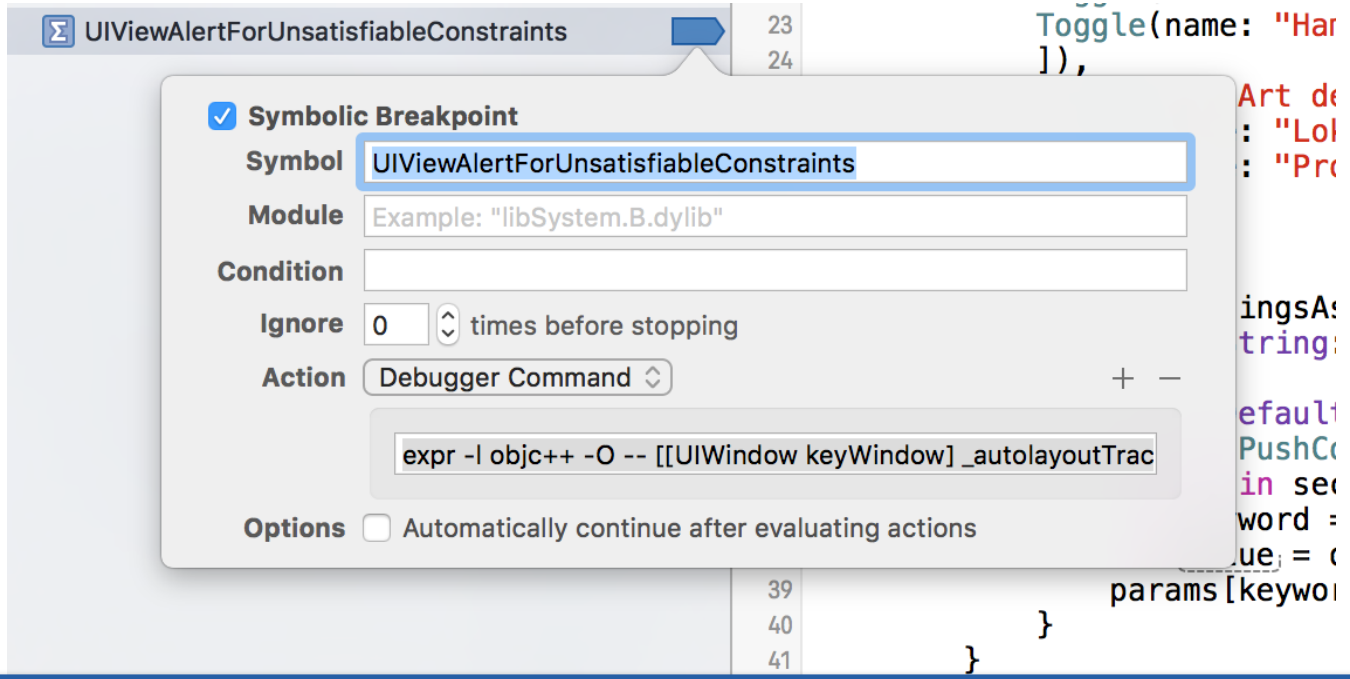
Demo
Size Classes / Adaptive Layout

Errors & Debugging

Autolayout Errors - WTF?

```
Unable to simultaneously satisfy constraints.
Probably at least one of the constraints in the following list is one you don't want.
Try this:

(1) look at each constraint and try to figure out which you don't expect;
(2) find the code that added the unwanted constraint or constraints and fix it.
(Note: If you're seeing NSAutoresizingMaskLayoutConstraints that you don't understand, refer to
the documentation for the UIView property translatesAutoresizingMaskIntoConstraints)

(...........)


Make a symbolic breakpoint at UIViewAlertForUnsatisfiableConstraints to catch this in the
debugger.
The methods in the UIConstraintBasedLayoutDebugging category on UIView listed in <UIKit/UIView.h>
may also be helpful.
```

Auto Layout Breakpoints - FTW!




```
Symbol:
UIViewAlertForUnsatisfiableConstraints
```

```
Action:
expr -l objc++ -O -- [[UIWindow keyWindow] _autolayoutTrace]
```

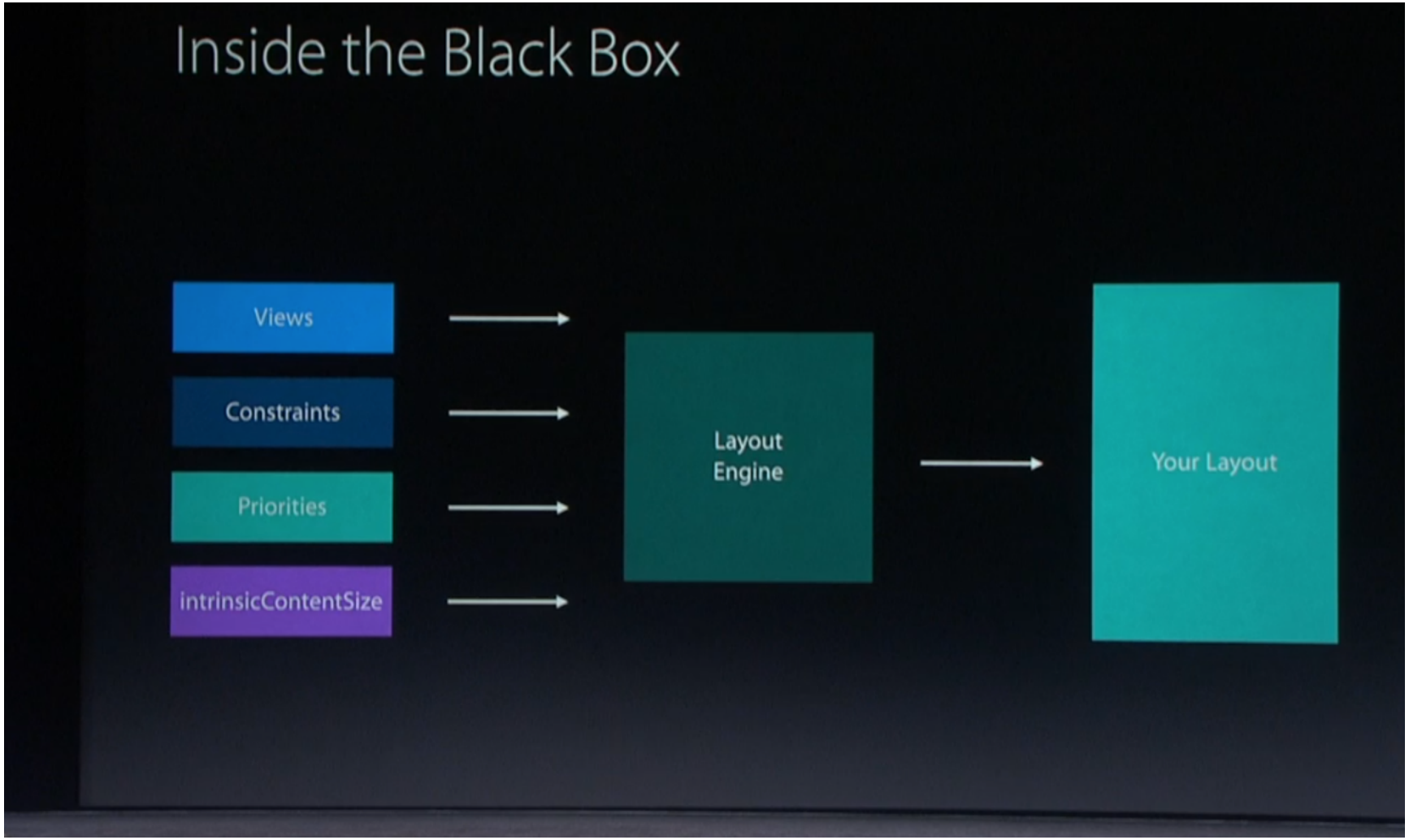
Auto Layout Breakpoints

- gibt uns -> gute Konsolen-Ausgaben

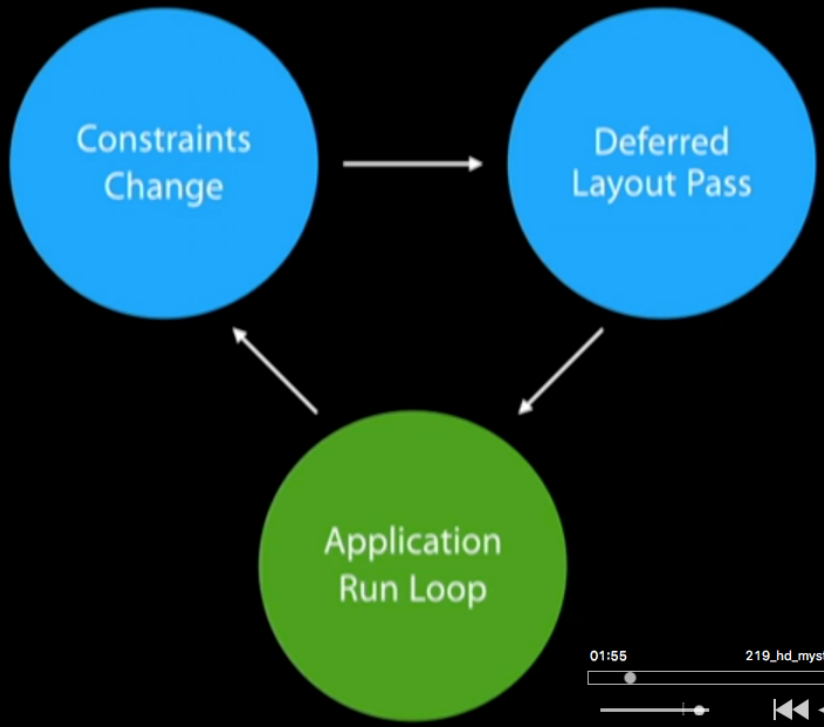
```
UIWindow:0x7f9481c93360
|   •UIView:0x7f9481c9d680
|   |   *UIView:0x7f9481c9d990- AMBIGUOUS LAYOUT for UIView:0x7f9481c9d990.minX{id: 13}, UIView:
0x7f9481c9d990.minY{id: 16}
|   |   *_UILayoutGuide:0x7f9481c9e160- AMBIGUOUS LAYOUT for _UILayoutGuide:0x7f9481c9e160.minY{id: 17}
|   |   *_UILayoutGuide:0x7f9481c9ebb0- AMBIGUOUS LAYOUT for _UILayoutGuide:0x7f9481c9ebb0.minY{id: 27}
```

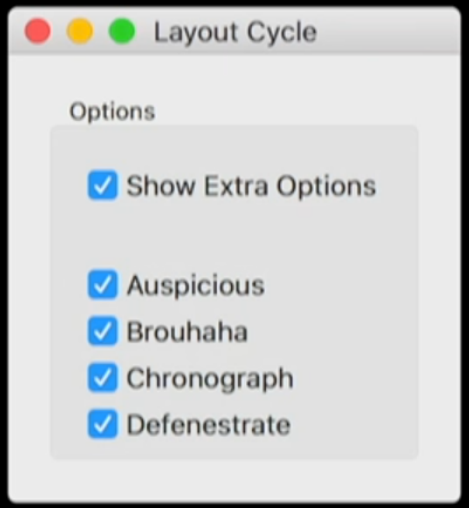
- die wir dennoch nicht wirklich verstehen 🤔
- die Lösung: Hintergrundfarbe des Übeltäters ändern 🤓

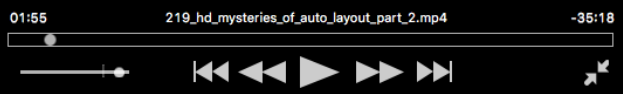
```
(lldb) expr ((UIView *)0x7f9ea3d43410).backgroundColor = [UIColor redColor]
(UICachedDeviceRGBColor *) $1 = 0x00007f9ea3d43410
```


Inside the Black Box
Views
Constraints
Priorities
intrinsicContentSize
Layout Engine
Your Layout

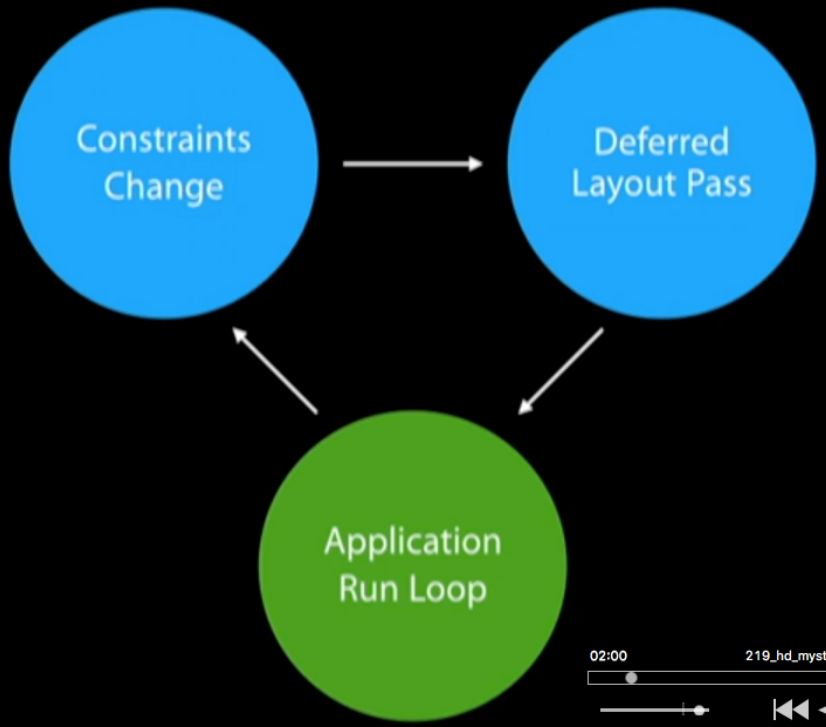
The Layout Cycle

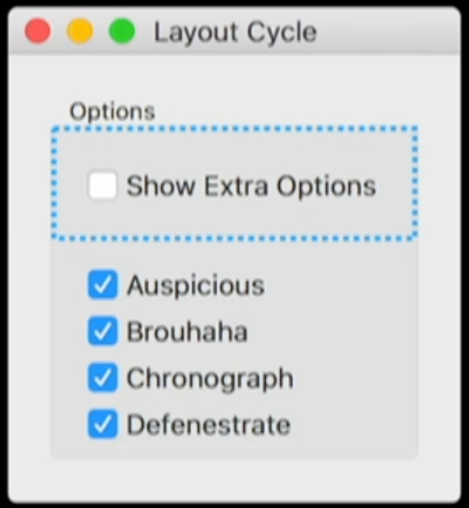


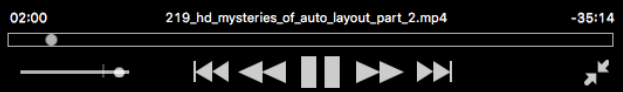




The Layout Cycle







Resolving Ambiguity

Diagnostic tools

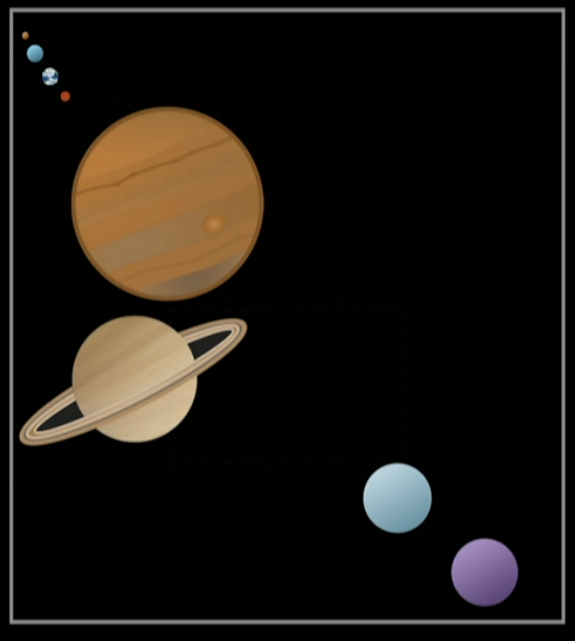
Red and yellow icons in IB

`_autolayoutTrace`

Select Debug > View Debugging

Look in the view debugger

`exerciseAmbiguityInLayout`



Resolving Ambiguity

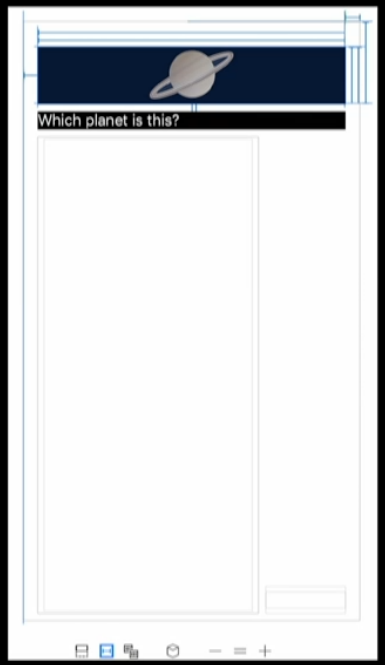
Diagnostic tools

Red and yellow icons in IB

`_autolayoutTrace`

Select Debug > View Debugging

Look in the view debugger



Resolving Ambiguity

Diagnostic tools

Red and yellow icons in IB

`_autolayoutTrace`

Select Debug > View Debugging

Look in the view debugger

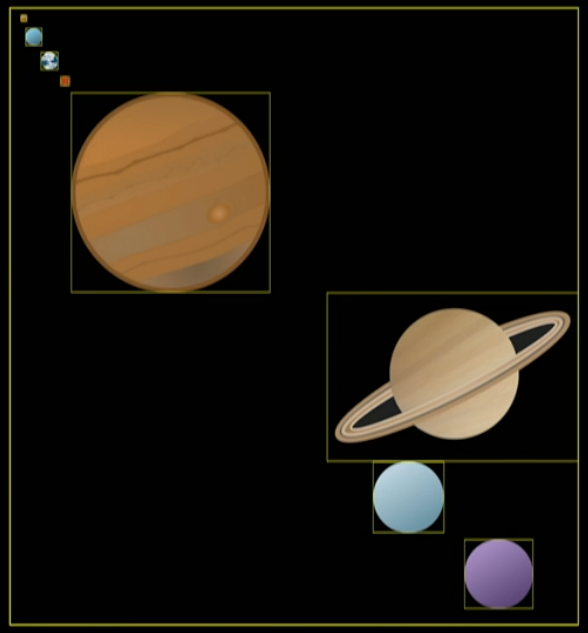
Resolving Ambiguity

Diagnostic tools

Red and yellow icons in IB

`_autolayoutTrace`

Select Debug > View Debugging



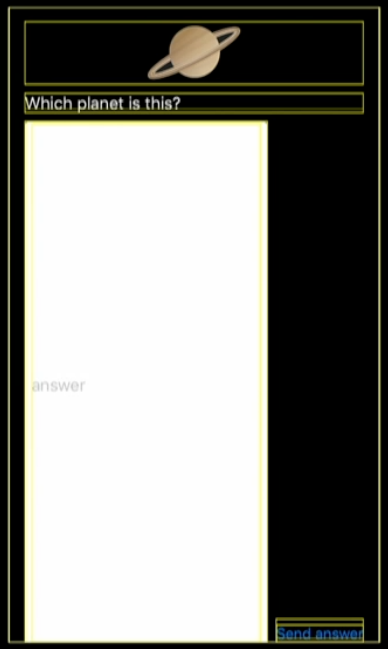
Resolving Ambiguity

Diagnostic tools

Red and yellow icons in IB

`_autolayoutTrace`

Select Debug > View Debugging




esolving Ambiguity
iagnostic tools
ed and yellow icons in IB
autolayoutTrace
elect Debug > View Debugging
Take Screenshot of Active Device
Capture View Hierarchy
Show View Frames
Show Alignment Rectangles
Show View Drawing
Show Responsive Scrolling Status
Show Focusable Regions

Resolving Ambiguity

Diagnostic tools

Red and yellow icons in IB

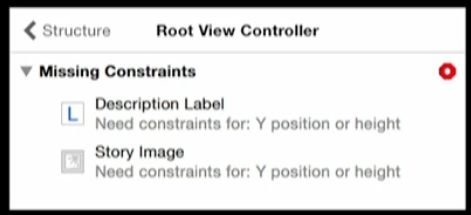
`_autolayoutTrace`

```
(lldb) po [self.view _autolayoutTrace]

UIWindow:0x7fe7434a3fe0
|   •UIView:0x7fe7434a8140
|   |   *_UILayoutGuide:0x7fe7434a84f0
|   |   *_UILayoutGuide:0x7fe7434a90d0
|   |   *Mercury:0x7fe7434a7790
|   |   *Venus:0x7fe743639380
|   |   *Earth:0x7fe74363aae0
|   |   *Mars:0x7fe74363bed0
|   |   *Jupiter:0x7fe74363ce30
|   |   *Saturn:0x7fe74363e220- AMBIGUOUS LAYOUT for Saturn.minX{id: 165}
|   |   *Uranus:0x7fe74363f690
|   |   *Neptune:0x7fe743640d60

Legend:
    * - is laid out with auto layout
    + - is laid out manually, but is represented in the layout engine
because translatesAutoresizingMaskIntoConstraints = YES
    • - layout engine host

(lldb)
```

Resolving Ambiguity

Diagnostic tools

Red and yellow icons in IB

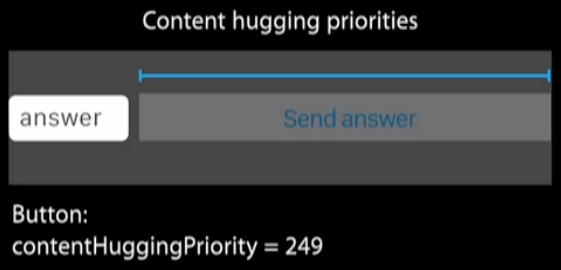


Ambiguous Layouts

Why doesn't my layout look right?

Possible causes

- Too few constraints
- Conflicting priorities



Understanding the Log

Start from the bottom

Check `translatesAutoresizingMaskIntoConstraints`

Set identifiers

Use `constraintsAffectingLayoutForAxis:`

Understanding the Log

Adding identifiers

Use constraint identifiers

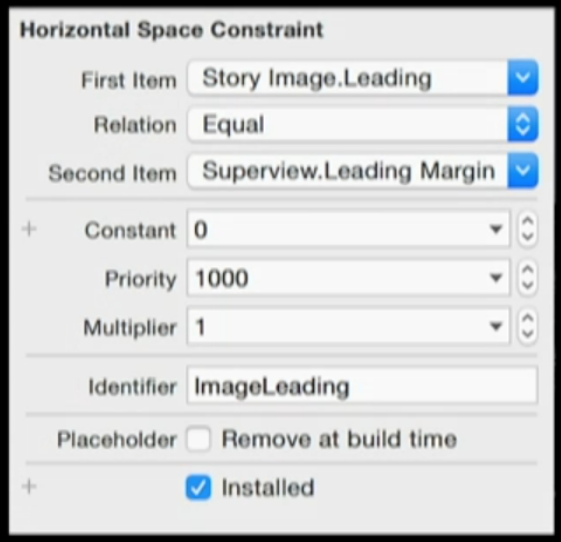
Explicit constraints

```
labelToTop.identifier = @"labelToTop";
```

Constraints using VFL

```
for (NSLayoutConstraint *constraint in verticalLayout)
{
    constraint.identifier = @"verticalLayout";
}
```

Constraints in Interface Builder



Understanding the Log

```
(
    "<_UILayoutSupportConstraint:0x7ffe9ad11a80 V:[_UILayoutGuide:
0x7ffe9ad10650(20)]>",
    "<_UILayoutSupportConstraint:0x7ffe9ad10ba0 V:|-(0)-[_UILayoutGuide:
0x7ffe9ad10650]   (Names: '|':UIView:0x7ffe9c81b720 )>",
    "<NSLayoutConstraint:0x7ffe9acbef60 'saturnWidth ' saturn.width ==
1.5*saturn.height   (Names: saturn:0x7ffe9acb8cb0 )>",
    "<NSLayoutConstraint:0x7ffe9c905460 'imageHorizontal' saturn.leading
== UIView:0x7ffe9c81b720.leadingMargin   (Names: saturn:
0x7ffe9acb8cb0 )>",
    "<NSLayoutConstraint:0x7fedd3423ae0 'imageHorizontal' UIView:
0x7fedd3607b90.trailingMargin == saturn.trailing>",   (Names: saturn:
0x7ffe9acb8cb0 )>",
    "<NSLayoutConstraint:0x7ffe9c905aa0 'verticalLayout' V:
[_UILayoutGuide:0x7ffe9ad10650]-(NSSpace(8))-[saturn]   (Names: saturn:
0x7ffe9acb8cb0 )>",
    "<NSLayoutConstraint:0x7ffe9c905b40 'verticalLayout' V:[saturn]-
(NSSpace(8))-[UILabel:0x7ffe9c903d10'Which planet is this?']   (Names:
saturn:0x7ffe9acb8cb0 )>",
    "<NSLayoutConstraint:0x7ffe9c906050 'labelToTop' V:|-(100)-[UILabel:
0x7ffe9c903d10'Which planet is this?']   (Names: '|':UIView:0x7ffe9c81b720
)>",
    "<NSLayoutConstraint:0x7ffe9aca0130 'UIView-Encapsulated-Layout-Width'
H:[UIView:0x7ffe9c81b720(375)]>"
)

Will attempt to recover by breaking constraint
<NSLayoutConstraint:0x7ffe9acbef60 'saturnWidth ' saturn.width ==
1.5*saturn.height   (Names: saturn:0x7ffe9acb8cb0 )>
```

NSLayoutGuide / UILayoutGuide

NEW

`UILayoutGuide` represents a rectangle in the layout engine

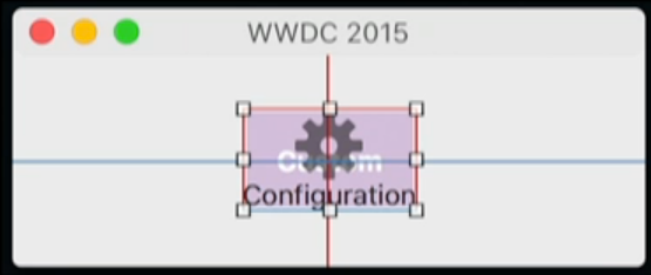
Constrain just like a view

```
let guide = UILayoutGuide()
view.addLayoutGuide(guide)
```

Constraining Negative Space

Equal spacing between buttons

Centering a group



Layout Constraint Creation

NEW

Layout anchors

```
NSLayoutConstraint(item:b, attribute:.Top, relatedBy:.Equal, toItem:view,
   attribute:.Top, multiplier:1, constant:10)
NSLayoutConstraint(item:b, attribute:.Leading, relatedBy:.Equal,
   toItem:view, attribute:.Leading, multiplier:1, constant:10)
```

```
b.topAnchor.constraintEqualToAnchor(view.topAnchor, constant:10)
b.leadingAnchor.constraintEqualToAnchor(view.leadingAnchor, constant:10)
```

Layout Constraint Creation

```
NSLayoutConstraint(item:b, attribute:.Top, relatedBy:.Equal, toItem:view,
   attribute:.Top, multiplier:1, constant:10)
NSLayoutConstraint(item:b, attribute:.Leading, relatedBy:.Equal,
   toItem:view, attribute:.Leading, multiplier:1, constant:10)
```

translatesAutoresizingMaskIntoConstraints

Do not forget to turn this off!

```
override func viewDidLoad() {
    super.viewDidLoad()

    let b = NSButton()
    b.bezelStyle = .RoundedBezelStyle
    b.translatesAutoresizingMaskIntoConstraints = false
    view.addSubview(b)

    NSLayoutConstraint(item:b, attribute:.Top, relatedBy:.Equal, toItem:view,
       attribute:.Top, multiplier:1, constant:10).active = true
    NSLayoutConstraint(item:b, attribute:.Leading, relatedBy:.Equal,
       toItem:view, attribute:.Leading, multiplier:1, constant:
       10).active = true
}
```

translatesAutoresizingMaskIntoConstraints

Do not forget to turn this off!

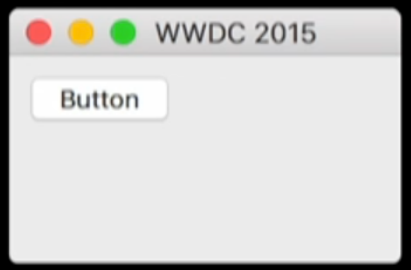
```
2015-05-08 09:41:27.668 WWDC 2015[4107:226949] Unable to simultaneously
satisfy constraints:
(
    "<NSAutoresizingMaskLayoutConstraint:0x6100000810e0 h=--& v=--& H:|-(0)-
[NSButton:0x618000140160'Button']   (Names: '|':NSView:0x618000120460 )>",
    "<NSLayoutConstraint:0x6180000828a0 H:|-(10)-[NSButton:
0x618000140160'Button'](LTR)   (Names: '|':NSView:0x618000120460 )>"
)
Will attempt to recover by breaking constraint
<NSLayoutConstraint:0x6000000825d0 H:|-(10)-[NSButton:0x600000140c60'Button']
(LTR)   (Names: '|':NSView:0x6000001203c0 )>
```

`translatesAutoresizingMaskIntoConstraints`

Do not forget to turn this off!

translatesAutoresizingMaskIntoConstraints

Do not forget to turn this off!




```
override func viewDidLoad() {
    super.viewDidLoad()

    let b = NSButton()
    b.bezelStyle = .RoundedBezelStyle

    view.addSubview(b)

    NSLayoutConstraint(item:b, attribute:.Top, relatedBy:.Equal, toItem:view,
       attribute:.Top, multiplier:1, constant:10).active = true
    NSLayoutConstraint(item:b, attribute:.Leading, relatedBy:.Equal,
       toItem:view, attribute:.Leading, multiplier:1, constant:
       10).active = true
}
```

translatesAutoresizingMaskIntoConstraints

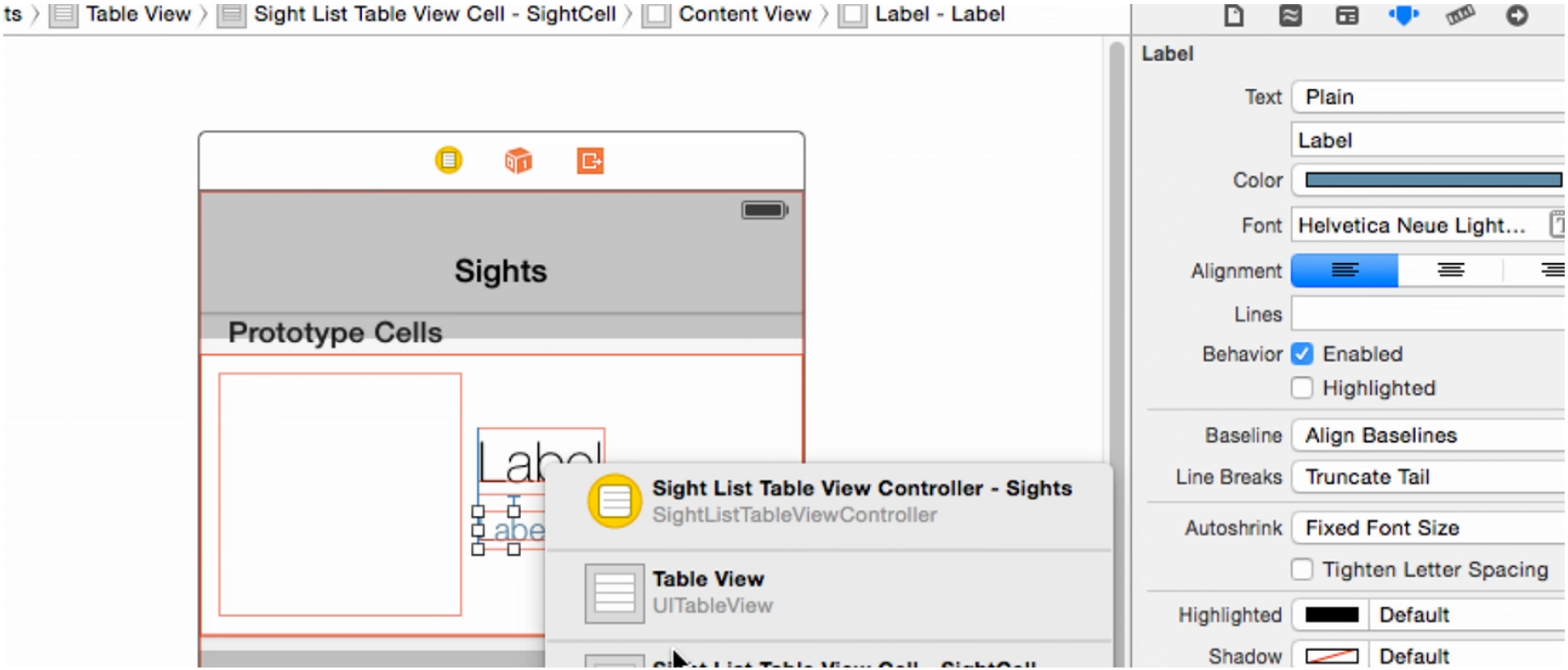
Setting the frame automatically generates constraints

- Set the frame with gleeful abandon!
- Constraints implement the autoresizingMask
- Other views can be constrained to it

Set to false when using constraints

- Beware–defaults to true for programmatically created views

Tipp: Shift


ts
Table View
Sight List Table View Cell - SightCell
Content View
Label - Label
Sights
Prototype Cells
Sight List Table View Controller - Sights
SightListTableViewController
Table View
UITableView
Label
Text
Plain
Label
Color
Font
Helvetica Neue Light...
Alignment
Lines
Behavior
Enabled
Highlighted
Baseline
Align Baselines
Line Breaks
Truncate Tail
Autoshrink
Fixed Font Size
Tighten Letter Spacing
Highlighted
Default
Shadow
Default

Look out for UILayoutGuide*

*Unfortunately Code only at the moment

UILayoutGuide

> IMPORTANT
>
> This is a preliminary document for an API or technology in development. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein for use on Apple-branded products. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future betas of the API or technology.

Inherits From

- NSObject
  - UILayoutGuide
    - UIFocusGuide

Conforms To

- CVarArgType
- CustomStringConvertible
- Equatable
- Hashable
- NSCoding
- NSObjectProtocol

Import Statement

SWIFT

```
import UIKit
```

Availability

Available in iOS 9.0 and later

The `UILayoutGuide` class defines a rectangular area that can interact with Auto Layout.

https://developer.apple.com/library/prerelease/ios/documentation/UIKit/Reference/UILayoutGuide_Class_Reference/index.html

“Really learn Interface Builder
This makes prototyping much faster”

Chris Eidhof